# Energy Efficient Software Development

**By Tyler McDonald**


## Abstract

There has been a lot of discussion over the past few years on climate change, and what we can do to lessen our impact on the environment. One of the largest concerns is that people are simply using too much energy. Many researchers in the industry have been looking for ways to reduce energy consumption in the technology that we use daily. Much of this research focuses on making hardware components more efficient. However, systems can also benefit from making the software efficient as well. In this paper, we will investigate several different methods that can be used for developing energy efficient software. The methods are not necessarily limited to programming. There are also ways to consider energy efficiency in the design phase of software development that will be investigated as well.


## 1. Introduction

Energy consumption has a large impact on the environment that we live in. Information and Communications Technology (ICT) systems are a large contributor to that. In fact, ICT systems are responsible for the same amount of $CO_2$ emissions as global air travel, which accounts for 2% of global $CO_2$ emissions [4]. Energy consumption also has a large impact on the wallets of the people that have to pay for it. Therefore, it makes sense to try to do as much as we can to lessen these environmental and fiscal costs. This is not a new point being brought up now, as this has been a subject of research for decades. The need for energy efficient technology is only increasing as time moves forward, as technology continues to rapidly expand. For a good example of this expansion, consider the fact that over 15 years, the global number of existing web servers rose from 376 thousand to 395 million [4]. Considering that these numbers are ten years old, the number of existing web servers now is likely even larger.

Although there has been heavy research on the subject of energy efficiency in the ICT field, most of this research focuses on making hardware components more efficient. This is absolutely helpful, but in many systems, the behavior of hardware components is controlled by software. Because of this, if the software is not efficient, then the hardware it controls is not being utilized fully, and the gains in efficiency for the hardware are effectively lost. To see the best results, both software and hardware should be improved as much as possible, so that the entire system is efficient.

For the purposes of this paper, three major sources on the topic of energy efficiency in software were investigated. The sources each contain their own unique viewpoints on how the goal of creating energy efficient software can be accomplished. Each of these sources will be summarized here.

## 2. Energy Efficient Programming

The first of the major sources to be summarized is simply titled 'Energy Efficient Programming.' It is a huge paper that covers a lot of ground, but it does not necessarily go in depth on any topic it brings up. For this reason it is a good starting point, as it introduces the reader to many different topics for further research. For the purposes of this paper, we will focus on what is relevant here, and leave the rest out.

### 2.1. Introduction

Energy efficiency and computing performance have basically been growing at the same rate, effectively canceling out improvements on the efficiency side. Since the amount of computers in the world has been doubling every three years on average since 2008, the need for efficiency becomes bigger and bigger. As mentioned earlier, the energy consumption of ICT systems impacts our environment. The heat generated by the systems causes a demand for better thermal management, which impacts costs as well. The current annual power and cooling costs of servers represent about 60% of a servers acquisition cost [1].

### 2.2. Foundations

One of the main motivations for anything in this world is money. This is, of course, a factor in this research as well. In 2007, ICT systems were responsible for 2% of the global power consumption, which is equivalent to the annual power production of eight nuclear plants [1]. Ten years later, this number has likely grown, and it will continue to grow for the foreseeable future. Obviously, with that much power consumption, the cost is going to be enormous. Thirty percent of a data center's expenses are related to energy [1]. As power consumption increases, so too does the complexity of the design for power sources, which of course will also raise costs.

There are also environmental aspects to be considered as well. The author goes into detail on many of the different effects that global warming can have on the environment, but as this is not the subject of our research, this will be excluded here. That is not to say that they aren't important, but there are much better sources for the effects of global warming than a paper on software efficiency.

### 2.3. Measuring Energy Efficiency

In order to talk about energy efficiency, it is important to strictly define what that actually means in concrete terms. As software systems are extremely diverse, there is currently no standard way to define efficiency for software. Typically, efficiency is defined as the useful work done divided by the effort required. However, as stated, software's diversity makes it difficult to set a standard for what work done even means. For example, how would one measure the work done by a text editor, and compare that to the work done by a web browser? This

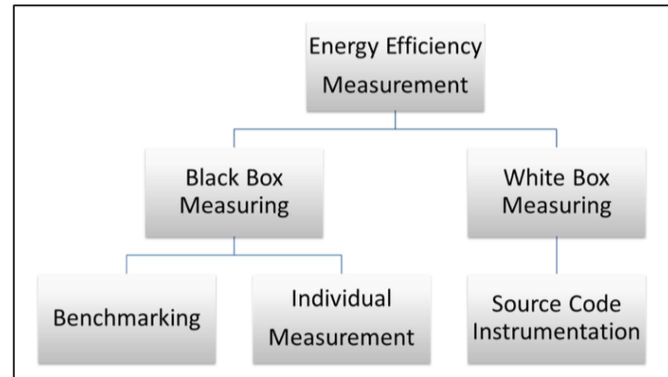


Figure 1: Methods for measuring energy efficiency

makes it clear that we will need different methods for different types of software, as there is no 'one size fits all' here.

As seen in Figure 1, the author has provided a breakdown of some of the important methods for measuring energy efficiency in software. We will discuss each of them here.

### 2.3.1. Black Box Measurement

Black box measurement effectively treats the entire software system as one thing, ignoring all of the components that actually make up the system. With this measurement method, we are only interested in the software as a whole, which means that should any problems be found, we will need a more sophisticated method to find where they are occurring within the system.

One of the methods falling under this category is known as benchmarking. This involves simply running the software against a standard set of tests. While the tests are running, energy consumption is measured. There are several different benchmarks that can be used for measuring energy efficiency, such as EnergyBench.

The other method falling under this category is individual measurement. This process involves coming up with several specific use cases, and measuring the performance of different configurations or types of software by running through the scenarios. These performance measurements can then be compared.

### 2.3.2. White Box Measurement

Unlike black box measurement, the white box methods look at software for what it really is: a complex system of different modules interacting together. These methods allow you to pinpoint where deficiencies in energy efficiency lie. However, they require a greater understanding of the system being measured, as the tester must interact with the code itself.

The white box method mentioned by the author is called source code instrumentation. This involves inserting instructions into the code itself meant to monitor different aspects of the system. It is much more difficult to implement than the black box methods, and because of this, there are not many examples that can be given.

### 2.4. Energy Efficient Programming Methodologies and Common Problems

This section is basically the meat of the paper, and discusses a lot of different methods to assist in developing energy efficient software. The methods are broken up into three categories: application software, system software, and general.

#### 2.4.1. Application Software Efficiency

In this section the author lays out many different methods for improving the efficiency of application software. Application software is the software that is designed for end users. Examples include web browsers, text editors, etc. These methods should be kept in mind when developing such software.

##### 2.4.1.1. Computational Efficiency

These methods are pretty general, and can be applied to most software development. Basically the goal here is to get the task at hand done as quickly as possible in order to get the computer back to an idle state. The computer consumes the most energy when it is actually working, so the quicker we can get back to idle, the better.

The first method brought up is using the best, most efficient algorithms for the task at hand. There are always multiple ways to go about solving a problem, and obviously some ways are faster than others. We want to pick the fastest method available. For example, there are many different sorting algorithms available for use in different situations. In an experiment conducted, 200,000 double values were sorted using two popular sorting algorithms known as bubble sort and heap sort. In order to complete the same task, bubble sort consumed 10,800 Joules, while heap sort only consumed 7,325 Joules [1]. This demonstrates that you should always analyze the running times of the available algorithms and determine which one fits your needs best. This idea also applies to the data structures you choose. For example, if the most frequent operations you are doing with stored data are insertion and deletion, you would want to stay away from a tree, as those operations increase in running time depending on the size of the tree.

Another way to increase the energy efficiency of a program is to be careful with loops. Loops can cause a lot of overhead, since on each iteration of a loop a comparison must be made to determine if it is time to halt the loop and move on. There are also often counters involved with this process, which means we have the added operation of adding or subtracting from this

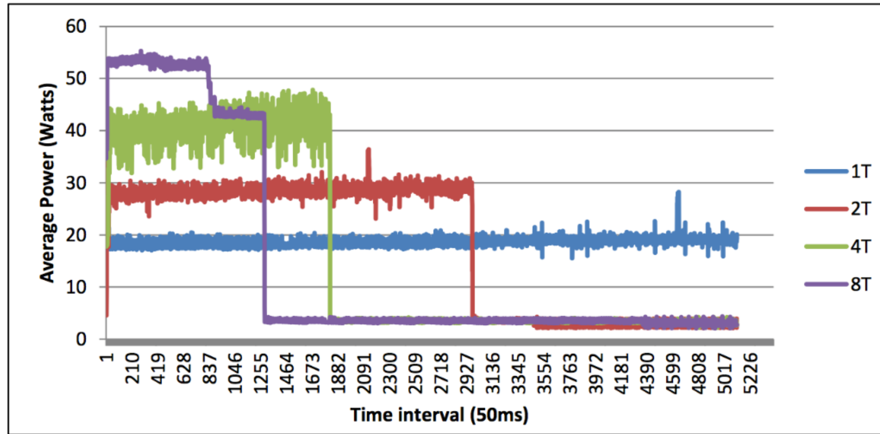


Figure 2: Impact of multithreading on energy consumption

counter. In some situations it can be beneficial to get rid of the loop entirely, and simply repeat the instructions that would have been executed individually to get rid of this overhead. Although this may look ugly, it can help.

A third method for reducing energy consumption is to utilize multi-threaded approaches where you can. For a task that does not require all of its steps to be done in sequence, many times parts of the task can be split and done simultaneously. Although this can be difficult to manage, it obviously has the benefit of getting the work done faster, which meets the goal of increasing idle time. Figure 2 shows the benefits of this approach. As you can see, increasing the number of threads causes an increase in the power consumption at the start, but since the task is completed much quicker, the power drops off to idle much faster.

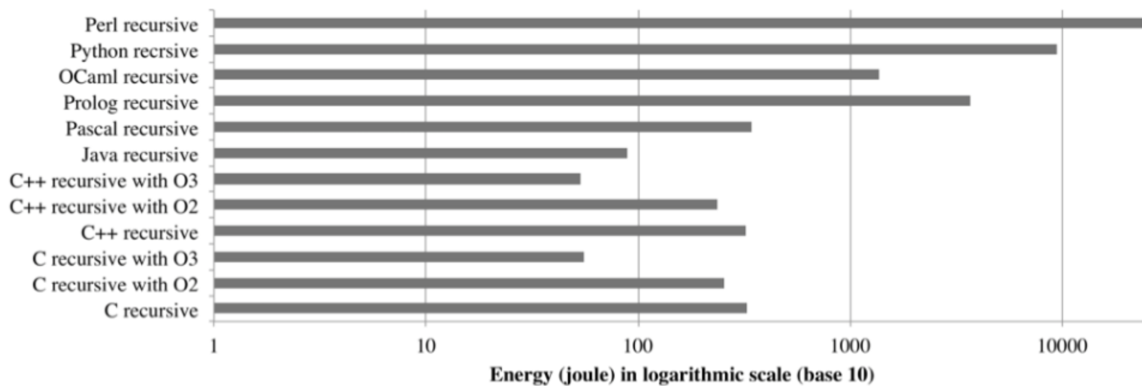


Figure 3: Energy consumption for different languages

Another consideration to make is whether or not there is already an existing library that can handle your task. Many software developers prefer to write their own code for everything, but often enough, there is already an excellent solution that has been proven to be energy efficient, saving you the effort.

Finally, choosing the right programming language for your task can make a huge difference as well. All languages handle things differently, meaning that energy consumption can vary drastically depending on which language a system is implemented in. This may require expert knowledge of the ins and outs of different languages, but it has been shown to have a dramatic effect on energy consumption. Figure 3 shows the results of an experiment running the same recursive algorithm in different languages. It is clear that in this case, the best language is C++, but this could simply be owed to how C++ handles function calls. In other cases, there will obviously be different results.

#### 2.4.1.2. Data Efficiency

This section deals with methods to improve the efficiency of data movement. The basic idea here is to complete a task with as few memory accesses as possible, and moving data over as short a distance as possible.

The closer data is stored to the processor, the less energy is consumed when accessing that data. This basically means that you should try to store data as close to the processor as possible. This can be accomplished by utilizing caching methods. With the same energy to access external RAM once, the computer can execute 7 instructions or access cache 40 times or access Tightly Coupled Memory (TCM) around 170 times [1]. There is plenty of very detailed research on caching available if the reader wishes to go further into this topic.

### 2.4.2. Operating Systems

This section focuses on methods to keep in mind when designing system level software, or operating systems. For the purposes of this paper we will not go into depth on this section, as most developers will never touch an operating system themselves. However, it is important to keep your target environment in mind when developing software. For example, it may be more efficient to run a web server on a Linux machine rather than a Windows machine. The developer should investigate the power consumption of their software in all available environments before deciding the best alternative.

### 2.4.3. General Problems and Solution Proposals

One of the most basic problems facing a software developer concerned with energy efficiency is the lack of focus on this problem. Most current software development models do not place any emphasis on energy consumption. It is not enough to focus on energy efficiency only in the implementation phase of development. The entire system should be designed with energy in mind. In other words, energy efficiency should be a must have non-functional requirement.

There is currently a model that can be referred to for energy efficient development, known as the GREENSOFT model. The GREENSOFT Model is a conceptual reference model for green and sustainable software that includes a product life cycle model for software products, sustainability metrics and criteria for software, software engineering extensions for sustainably sound software design and development, as well as appropriate guidance [1]. Unfortunately, most of the reference material for information on this model is locked behind a paywall.

### 2.5. Tools and Technologies

The author presents many different existing tools that can be used to help develop energy efficient software. Examples include PowerEscape, which is a tool with many different functions for increasing the data efficiency of your software. Another tool is called the Intel Web APIs, which can provide you with information about the platform that is executing your web service, allowing you to modify your approach appropriately. A third example is PowerInformer, which provides basic power statistics. Of course there are many other tools available, and the author cannot provide an exhaustive list.

## 3. Extending Software Architecture Views with an Energy Consumption Perspective

This paper proposes a perspective on software architecture where the architect tries to structure the software in the design phase of software development so that it will consume less energy. It is much more in depth than the previous source, and includes a case study.

### 3.1. Introduction

There has been too much focus on the hardware aspects of energy efficiency in research. While this is helpful, software also plays an important role. While energy is directly consumed by hardware, the operations are directed by software and can eliminate any sustainable features built into the hardware [2]. The author states that this makes software the 'true consumer of power.' A decrease in energy consumption of 0.25 watts for a software product that has four million installations saves the energy equivalent of the monthly power consumption for an American household [2].

As mentioned before, looking at energy consumption only in the implementation phase of software development is not enough. Energy should be considered at all phases of software development. Also, treating software as a single object, or black box, is not enough. All of the components of the software must be investigated. For these reasons, the architecture of software must be considered. Using an architecture description of software together with energy measurements can help direct efforts to reduce energy consumption in software.

### 3.2. Related Work

Here the author discusses the problem with energy consumption measurements. Although there are existing ways to measure the energy consumption of software, these methods require that a specialized environment be set up with extra equipment. Many are not willing to invest in that. It is also difficult or impossible to expand them to more complex environments such as data centers, or software that is distributed across multiple servers. Other methods investigating the code itself are also difficult to implement, as they require expert knowledge on the subject under study. So, if we could design software to be efficient from the start, these methods would not be required.

### 3.3. Sustainability as a Quality Attribute

Here the author sets out to define some concrete properties that can be considered when looking at software architecture. As seen in Figure 4, the main focus here is resource consumption. The author defines three quality properties as children of resource consumption. These properties are software utilization, workload energy, and energy usage. Software utilization is defined as the degree to which resources specifically utilized on the account of a software product meet requirements. Energy usage is the degree to which the amount of energy used by a software product meets requirements. Finally, energy usage is the degree to which the energy consumption related to performing a specific task using a software product meets requirements. There are many children for each of these properties, as can be seen in Figure 5. They each have their own formula associated with them.

Of course there will be trade offs between different priorities if energy consumption is added as a consideration. For example, you may want to implement some logging for security purposes. However, the extra task of logging will negatively impact the energy consumption. Now, since these attributes and properties are made concrete with measurements, a proper trade off analysis can be made, instead of just guessing.

### 3.4. Energy Consumption Perspective on Software Architecture

An architectural perspective is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the systems architectural views [2]. Here the author proposes

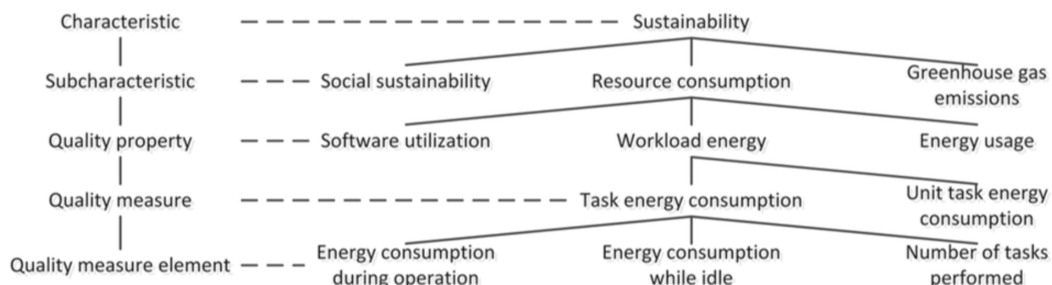


Figure 4: Breakdown of the sustainability characteristic

| Resource consumption | |
|---|---|
| **Software utilization** | |
| CPU Utilization (CPUU) | Measure of the CPU load related to running the software. *current CPU load − idle CPU load* |
| Memory Utilization (MU) | Measure of the memory usage related to running the software. $\frac{allocated\ memory}{total\ memory} \times 100\ \%$, *working memory, Private bytes, Virtual bytes* |
| Network Throughput (NT) | Measure of the network load related to running the software. *Packages per second, sent bytes per second, received bytes per second* |
| Disk Throughput (DT) | Measure of the disk usage induced by running the software. *Disk I/O per second* |
| **Energy usage** | |
| Software Energy Consumption (SEC) | Measure for the total energy consumed by the software. *EC while operating—idle EC* |
| Unit Energy Consumption (UEC) | Measure for the energy consumed by a specific unit of the software. $\left(\frac{Unit\ CPUU}{CPUU} \times \frac{Unit\ MU}{MU} \times \frac{Unit\ NT}{NT} \times \frac{Unit\ DT}{DT}\right) \times SEC$ |
| Relative Unit Energy Consumption (RUEC) | Measure for the energy consumed by a specific unit compared to the entire software instance. $\frac{UEC}{SEC} \times 100\ \%$ |
| **Workload energy** | |
| Task energy consumption (TEC) | Measure for the energy consumed when a task is performed. $\frac{SEC}{\#\text{ of tasks performed}}$ |
| Unit task energy consumption (UTEC) | Measure for the energy consumed when a task is performed by a specific unit of the software. $\frac{UEC}{\#\text{ of tasks performed}}$ |

Figure 5: Quality properties and measurements for resource consumption

an architectural perspective for energy consumption. Perspectives are meant to help assist an architect with their tasks.

This perspective includes this set of key questions that should be asked when considering the architecture of the software:

- How can the software product architecture assist in achieving an organization's sustainability strategy?
- How can run-time aspects be fine-tuned to reduce EC?
- How can we measure the EC of the different nodes the software is executed on?
- Which processes run on what hardware?
- How do the functional elements map onto processes?
- What processes can be executed concurrently without increasing the resource consumption related to their coordination and control?
- How much energy does each function consume?
- How can the information flow be optimized to increase EE?
- What green algorithms can be applied to the software and where should they be applied?

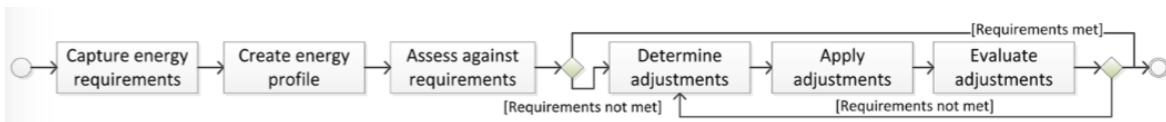


Figure 6: Flow of activities for perspective

Also included is a set of activities that should be done in order to apply the perspective in practice. These activities can be seen in Figure 6. The first step is to form the energy requirements for your product. Then you must create an energy profile for the product, which involves measuring the energy consumption. The next step is to assess the current energy profile against the requirements you made. If there are adjustments to be made, they should be found and applied. Once the adjustments are complete, you should determine whether or not the adjustments were successful. This process can include a loop if the changes were not successful.

The author also supplies a set of tactics that can be used to address any energy concerns for the product. These tactics are increasing modularity, optimizing network load, increasing hardware utilization, and concurrency architecture variation. They are fairly self explanatory, but a few can be explained with an example. For increasing modularity, think of a database. Having more modules can mean more calls to this database, which means that we are getting less data per call, and the calls are more fitted to the process at hand. This means that less CPU capacity has to be used for processing the calls, as they are smaller. In the case of network load optimization, however, you would want to do the opposite. By creating more database calls, you have increased the network load, as you are using the network more. This demonstrates the concept of trade offs.

### 3.5. Case Study: Applying the Perspective in Practice

For the case study that was performed, the author looked at Document Generator, a tool that is used to generate about 30 million documents per year. The author went through the

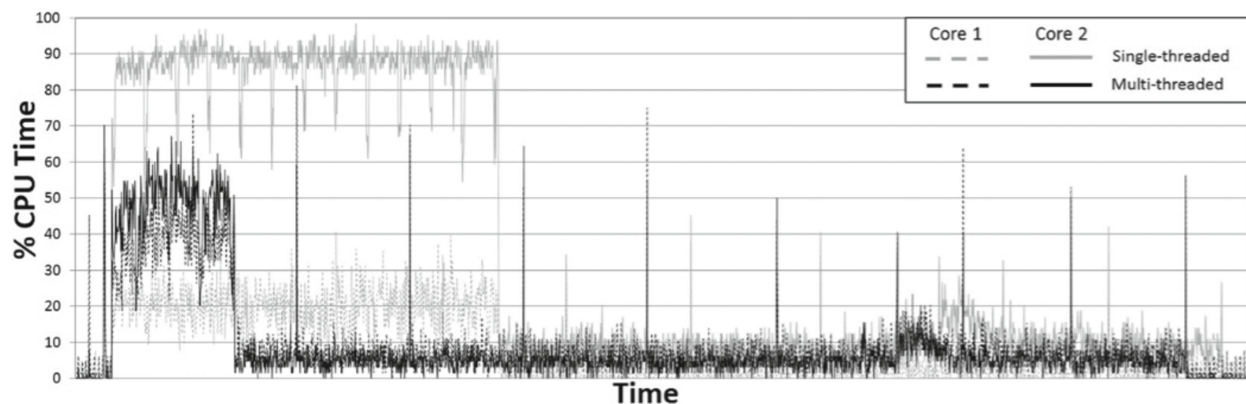


Figure 7: CPU Utilization before and after multithreading

process described in Figure 6 with the goal of reducing energy consumption. When an energy hotspot was found, they used the tactic of increasing hardware utilization. It was discovered that this hotspot was not multithreaded, and it was possible to do so. Figure 7 shows the results of this adjustment. As can be seen, CPU activity was reduced drastically. Applying this adjustment reduced the task energy consumption of generating the documents by 67.1%.

**4. ESUML-EAF**

This paper proposes a framework developed by the authors that can be used to create energy efficient design models for software. The framework is called Embedded Software modeling with UML 2.x - Energy Analysis Framework. The reason for this name will become clear at a later point. Again, this method is intended to be used in the design phase of software. It is also currently only intended for use in embedded software. The advantage of using a framework such as this is that it allows developers to fulfill the energy consumption requirements in the early phases of software design rather than later. This reduces the feedback that would have been caused had the requirements not been met later. Feedback is essentially the need to redo insufficient work from a previous phase.

**4.1. Introduction**

The field of embedded software is growing, and as such, the requirement for low energy is growing as well. Most existing studies in this area have focused on the hardware, but software has an impact too. The complexity and size of embedded software affects the energy consumption of the system that the software is embedded in. Fortunately, there has been some research on the software side as well.

The first energy consumption analysis technique was proposed in 1994. The basic idea is to break down the source code into low level instructions, measure the consumption of each of these instructions individually, and sum them. This led to new techniques being studied, and not all of them are instruction level. There are also source code based techniques, and even model based techniques. The issue with instruction and source code based techniques, however, is that although they are accurate, they require a lot of time to analyze. They also require even more time to go back and redo unsatisfactory results, because at this point the code is already written.

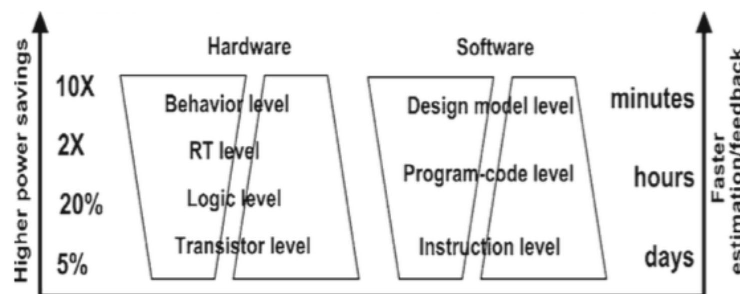


Figure 8: Energy analysis efficiency at different abstraction levels

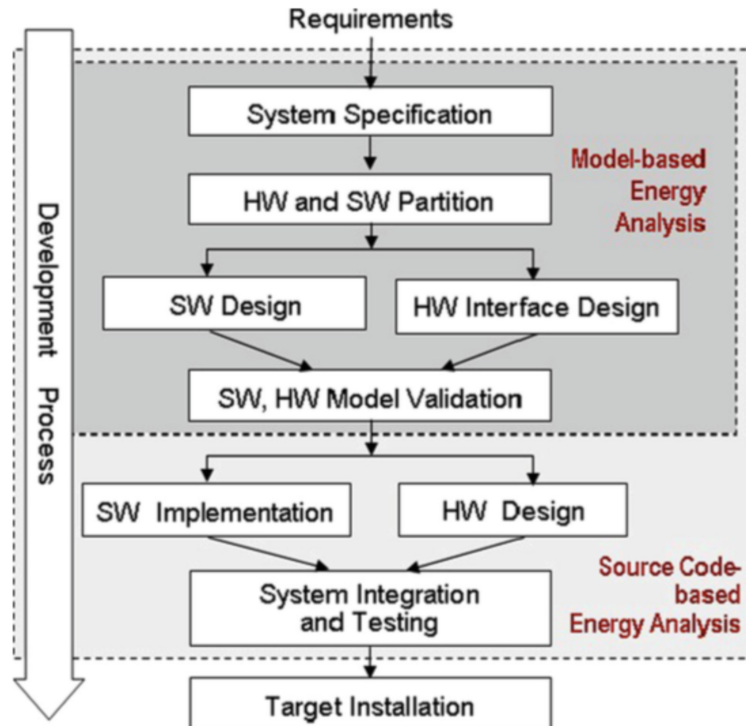


Figure 9: Energy analysis scope in software development

In contrast, model based techniques trade a small amount of accuracy for small analysis time. This is because the model is much more abstract than the source code itself. Another advantage is that the feedback for unsatisfactory results is cut down significantly, as there is no code to rewrite yet, just a model to adjust. Figure 8 shows the analysis and feedback time for the software analysis methods discussed, and Figure 9 shows there the analysis occurs in the software development cycle.

One of the drawbacks for model based methods is that they typically require an extra model for analysis. This requires extra work, and can deter people from trying the technique. The authors of this paper propose a framework that does not require any extra modeling to be done. This is because the framework is build to use models from UML, which should already exist if the developers followed a typical model-driven design process. The framework simply inspects the elements from the existing UML models, retrieves the energy consumption from the energy library also developed by the authors, and calculates energy consumption. This technique is intended to allow the developers to choose the model with the best energy efficiency from several alternatives.

**4.2. Related Work**

As mentioned before, there has been research on energy analysis techniques for software. These studies can be seen in Figure 10, grouped by the level of abstraction that they are based on. Instruction level techniques utilize an energy model that is constructed from actual

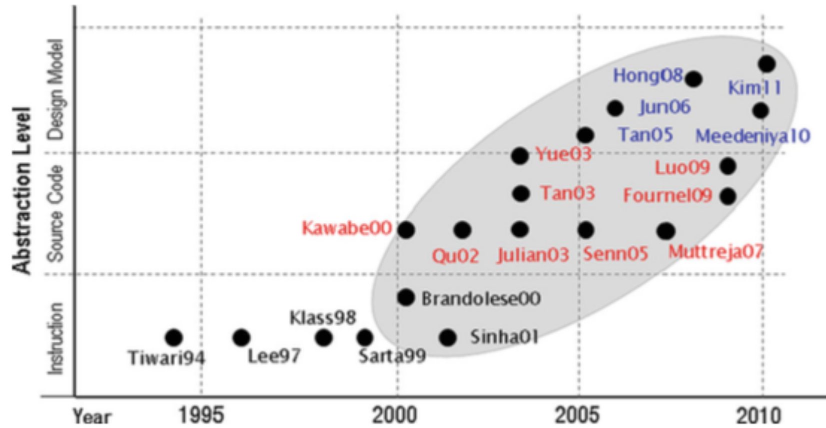


Figure 10: Studies grouped by abstraction level

measurement of energy consumption. This is acquired by actually executing or simulating each instruction, and can take days. Source code level techniques utilize higher level languages, such as C, to profile the code. This makes it faster than instruction level techniques, but it still takes a lot of time. This led researchers to search for a faster way to analyze energy consumption, and they came up with the model based techniques.

### 4.3. Framework Architecture

This section discusses the architecture of the framework that was developed. The overview can be seen in Figure 11. The major components are the ESUML modeler, CFG generator, ESUML energy library, energy realizer, and result viewer.

The ESUML modeler supports modeling using UML 2.0. The models it uses are the use case diagram, the class diagram, the interaction overview diagram, and the sequence diagram. The modeler also uses action language to describe the detailed behavior of execution

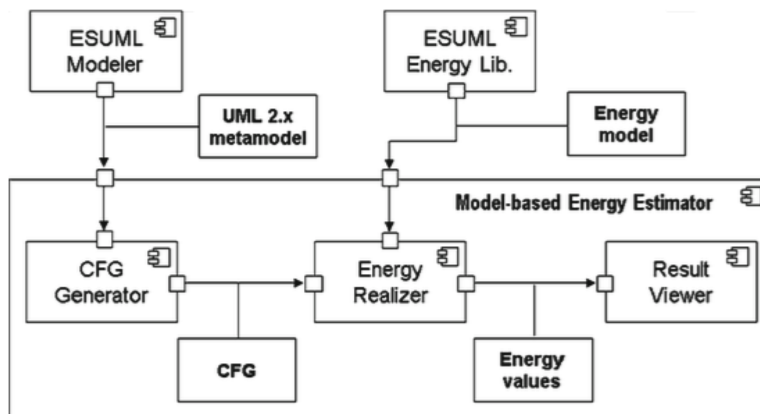


Figure 11: Overview of framework architecture

occurrences. The goal of the modeler is to represent the functional behavior of embedded software.

The Control Flow Graph (CFG) generator takes the UML models from the modeler as input. It basically transforms the models into a connected graph that represents the behavior of the software. It integrates all the models into one graph, and reveals synchronous action, asynchronous action, parallel action, branch action, and fork and join action.

The ESUML energy library is basically the core component of the framework. It contains all of the data required to analyze energy consumption. This data is organized by Energy Behavioral Units (EBUs). These will be explained in more detail later.

The energy realizer is essentially the component that does the actual calculations, taking the energy model and the control flow graph as input. It traverses the graph given, and calculates energy consumption values using the energy model.

The result viewer is simply a tool that can be used to view the results of this analysis. These results can be displayed as a total, by diagram, by EBU, or by class. The intent is to allow the user to determine where remodeling is needed.

**4.4. The Energy Library**

The energy library created for this framework is an improvement on some of the earlier work from the authors. One of the advantages for this energy library is that it is indexed by a more abstract element than most other energy libraries. Most use the instruction unit, whereas here EBUs are used. Each EBU is decomposed into a set of virtual instructions. Another advantage is that this library does not need to be changed if the software model is changed. The overall structure for the energy library can be seen in Figure 12.

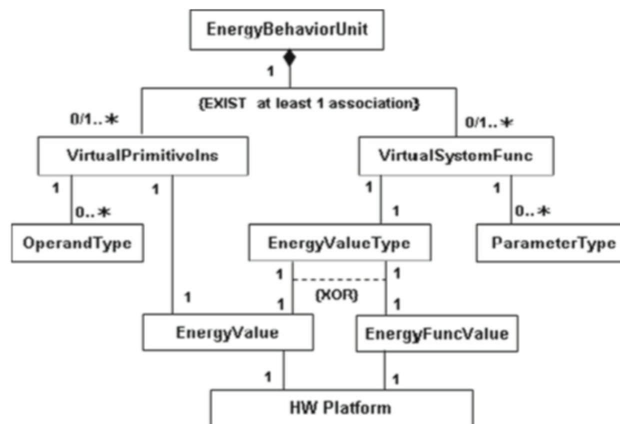


Figure 12: Structure of energy library

The first step used in building the energy library is EBU identification. This involves identifying the energy consuming elements that can be produced by the modeler. The elements come from the sequence diagrams, interaction overview diagrams, and action language. These elements are classified by type. The possible classifications are abstract, control structure, behavior execution, non-behavior, and action language. Of these types, abstract and non-behavior elements are discarded, as they have no influence on program execution.

The next step is to define the virtual instructions that can be used for each EBU. There are two different types of virtual instructions: virtual primitive instructions and virtual system functions. These virtual instructions are kept general in order to reduce dependency on hardware. The virtual instructions must then be measured to obtain energy values. This is done

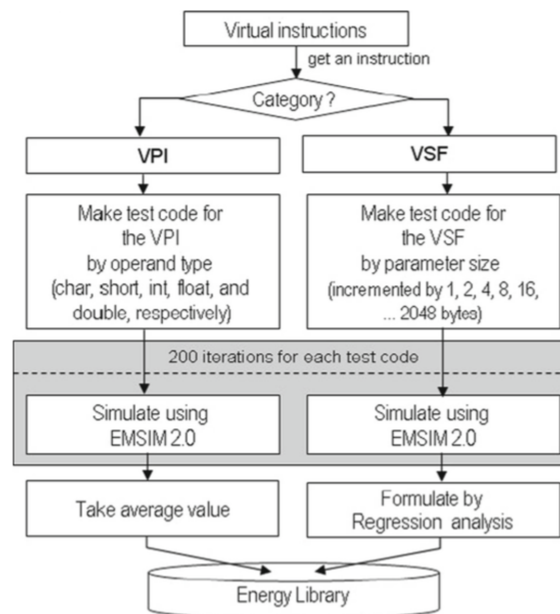


Figure 13: Virtual instruction profiling process

using a simulator. For each virtual instruction, the implementation code is simulated over 200 iterations for each operand type and parameter size. The energy consumption values for these simulations are then stored in the energy library. This process can be seen in Figure 13.

The final step is to map each EBU to the set of virtual instructions that it is represented by. This completes the mapping of EBUs to energy values, and the energy information for each EBU can be retrieved. The authors developed a simple tool with a GUI that can be used to manage this data.

**4.5 Experiment**

To test this framework, the authors looked at five algorithms that are regularly used in embedded systems. The algorithms are the shortest path selection algorithm for road navigation,

| Applications | Input data size (bytes) | Estimated energy consumption (nJ) | | Deviation (%) |
|---|---|---|---|---|
| | | Source-based | Model-based | |
| M1 | 288 | 136,982.1 | 135,419.3 | 1.15 |
| M2 | 32 | 2,237,200.5 | 2, 049,553.6 | 8.38 |
| M3 | 31,323 | 41,927,772.7 | 38,919,780.3 | 7.17 |
| M4 | 80 | 113,203.6 | 103,936.8 | 8.91 |
| M5 | 262,144 | 11,143,678.6 | 10,168,948.7 | 9.58 |

Figure 14: Comparison of analysis times

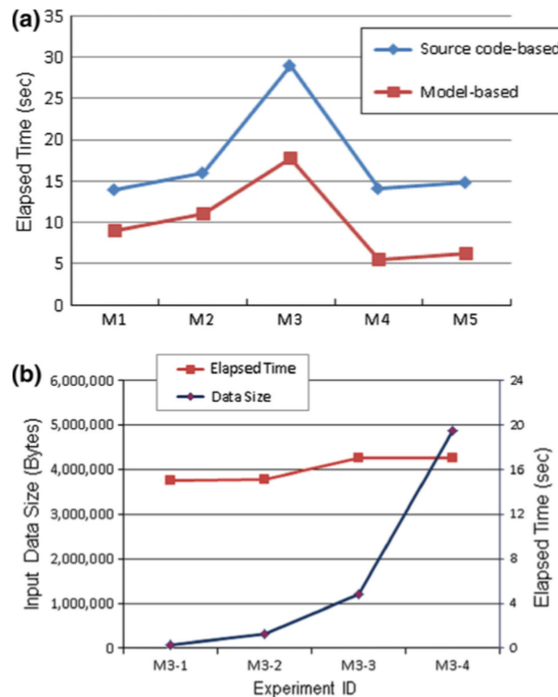


Figure 15: a) Comparison of analysis times
b) Elapsed time by data size for M3

an encryption algorithm using AES, and image encoding algorithm utilizing Huffman codes, the algorithm for data retrieval found in cell phones, and the image conversion algorithm found in digital cameras. These algorithms were each modeled using this framework. They were also each implemented by the authors as well, so that source code analysis time could be compared to model based. The results of this experiment are shown in Figures 14 and 15. As shown, the model based technique performs faster for every algorithm. Since we are only analyzing small algorithms rather than entire projects, the analysis time for source code is still relatively small, but the numbers for source code analysis would increase a lot faster than the numbers for model analysis. It is also shown that the elapsed time for model based analysis remains relatively stable. Figure 14 also shows that the deviation in energy consumption estimation between source code and model based techniques never exceeds ten percent. This means that the impact to accuracy is present, but negligible. Clearly this framework is an efficient way to choose the best possible design model for your product.

## 5. Conclusion

From these studies, we can safely conclude that software optimization is an excellent way to save energy. That is not to say that hardware should be ignored. Both software and hardware should be optimized for the best results. It is also apparent that although there are plenty of ways to reduce energy consumption by modifying the code, there also ways to improve your software before any code has even been written. All of the techniques presented in this paper can be used in conjunction with each other for excellent efficiency boosts overall. Although there has been some research on software optimization, it is still not a large focus in the industry. Work must be done to make industry leaders more aware of the possible environmental and fiscal benefits that can be received when technology is optimized correctly. Ideally this will lead to standard practices in the industry that are used by all developers.

## References


1. Goekkus and Konya. “Energy Efficient Programming.” 2013, files.ifi.uzh.ch/hilty/t/examples/bachelor/Energy_Efficient_Programming_Gökkus.pdf.
2. Jagroep, Erik, et al. "Extending Software Architecture Views with an Energy Consumption Perspective." Computing, no. 6, 2017, p. 553. EBSCOhost, doi:10.1007/s00607-016-0502-0.
3. Kim, Doo-Hwan and Jang-Eui Hong. "ESUML-EAF: A Framework to Develop an Energy-Efficient Design Model for Embedded Software." Software and Systems Modeling, no. 2, 2015, p. 795. EBSCOhost, doi:10.1007/s10270-013-0337-5.
4. Fettweis, Gerhard, and Ernesto Zimmermann. "ICT energy consumption-trends and challenges." Proceedings of the 11th international symposium on wireless personal multimedia communications. Vol. 2. No. 4. (Lapland, 2008.