*electronics*

MDPI

*Article*

# Elaborating Validation Scenarios Based on the Context Analysis and Combinatorial Method: Example of the Power-Efficiency Framework Innomterics

**Paolo Ciancarini [1,2,*], Artem Kruglov [2,*], Andrey Sadovykh [2], Giancarlo Succi [2,*] and Evgeniy Zuev [2]**

1 Department of Computer Science and Engineering, University of Bologna, 40138 Bologna, Italy
2 Institute of Software Development and Engineering, Innopolis University, 420500 Innopolis, Russia; a.sadovykh@innopolis.ru (A.S.); e.zuev@innopolis.ru (E.Z.)
* Correspondence: paolo.ciancarini@unibo.it (P.C.); a.kruglov@innopolis.ru (A.K.); g.succi@innopolis.ru (G.S.)

Received: 8 November 2020; Accepted: 8 December 2020; Published: 10 December 2020


**Abstract:** The preliminary task of a project consists of the definition of the scenarios that will guide further development work and validate the results. In this paper, we present an approach for the systematic generation of validation scenarios using a specifically developed taxonomy and combinatorial testing. We applied this approach to our research project for the development of the energy-efficiency evaluation framework named Innometrics. We described in detail all steps for taxonomy creation, generation of abstract validation scenarios, and identification of relevant industrial and academic case studies. We created the taxonomy of the target computer systems and then elaborated test cases using combinatorial testing. The classification criteria were the type of the system, its purpose, enabling hardware components and connectivity technologies, basic design patterns, programming language, and development lifecycle. The combinatorial testing results in 13 cases for one-way test coverage, which was considered enough to create a comprehensive test suite. We defined the case study for each particular scenario. These case studies represent the real industrial, educational, and open-source software development projects that will be used in further work on the Innometrics project.

**Keywords:** computer system taxonomy; programming language; development framework; design pattern; energy consumption; combinatorial testing; classification tree

## 1. Introduction

### *1.1. Rationale*

Nowadays, energy and mobility are essential aspects of the technological evolution of humankind. However, the global economy faces unprecedented challenges in meeting growing energy and mobility demands, due to the clash between economic development and resource limitations [1,2]. Every year, mobile device manufacturers seek to expand the range of devices. The new devices require more energy, one of the most urgent problems is to increase the number of hours of operation. An important issue to reduce the energy consumption of mobile devices, the ability of software components to adapt to their specific needs in order to minimize energy consumption [1].

The need to use energy-saving technologies is dictated not only by the desire to save resources, but also the inability to provide acceptable battery life for mobile devices. Today it is one of the driving forces behind the improvement of architectures and technologies such as mobile processors

and supercomputers and servers. Fully coping with the problem, using only hardware solutions (increase the battery capacity, optimization tools), is not possible, so we need to use software solutions and tools to assess, monitor, and predict the values of key parameters through the whole software development lifecycle.

An important role in the design of energy-efficient systems takes the analysis of values of the basic parameters (metrics) of energy consumption, like the utilization of CPU and GPU components. Software metrics are quantitative measures of specific software development artifacts, such as code complexity. There are process and product metrics, and both can be linked to energy consumption. Some of them based on the analysis of source code, developed during the past few decades for different programming paradigms such as structured programming and object-oriented programming (OOP) [3].

An important step in establishing a measurement program is the selection of the measures to use. The selection of the metrics should fit the development process and should have a direct impact on the quality of the delivered software. The set of metrics for different products or processes should be tailored even within the same organization. Metrics validation is another important topic in the area of software measurement because their acceptance depends on whether they are able to predict important qualities.

The use of a coherent system of metrics to evaluate energy consumption has been studied extensively in the literature [4,5]. The question of definition and application of metrics to evaluate the energy consumption is of great practical interest, and we have studied this issue in a companion paper recently published [6]. The system described in that paper—that we called InnoMetrics—aims at building and validating a quantitative framework to guide the development and the evolution of sustainable computer systems using a variety of metrics collected throughout the lifecycle of software products on different computer systems, from the initial concept to the deployment, execution, and maintenance, optimizing the performances of the products under a variety of nowadays relevant factors, including the efficient use of energy resources.

### *1.2. Motivation for the Research*

The technical tasks of our project, both theoretical and practical, have been conducted considering specific contexts. For these reasons, the preliminary task consists of the definition of a set of use cases and scenarios that will constitute the framework for guiding the technical work. Such usage scenarios are the major artifact in some agile frameworks for further exploration of the requirements for the development system [7].

There are several approaches to identify the scenarios, from an empirical approach based on the own level of expertise to the statistical methods, such as Monte Carlo sampling or First Two Moments sampling techniques [8]. Some research papers proposed the taxonomy of scenario development. Thus, Heugens and Oosterhout [9] provided a classification of the scenario generating methods based on the epistemology and normative involvement criteria. Van Notten [10] decomposed scenarios into several macro characteristics, such as goals, design process and content, and a number of detailed features to give a structural base for scenario generation. Bruninx [11] described several techniques for scenario generating based on scenario tree analysis: sampling, path-based methods, property matching, and probability metrics implementation.

For the current research, the primary idea for usage scenario generating is to specify a context in which the usage scenario will be executed, as a test case. In particular, it is important to identify examples of some technical infrastructure for which awareness of the status of resource usage and the viability of the system is critical (e.g., mobile devices, cloud computing, wireless sensor networks, etc.). Consequently, it will be possible to implement combinatorial testing techniques and tools for the selected parameters. Combinatorial methods will provide a reasonable number of scenarios with two-way or even three-way coverage that could be considered as an exhaustive analysis of the project deliverables. Based on this idea, the process of scenario generation consists of two parts:

- Create a classification tree for a given topic of energy-efficient software development. This process, including the selection of relevant aspects and identifying equivalence classes, is given in Section 2.
- Generate test cases. This process is given in Section 3.

Section 5 discusses the results in terms of generated test cases as illustrated by the generated samples for real case studies.

## 2. Classification Tree Development

In order to elicit appropriate scenarios for further analysis of the project results, we have to figure out which factors in software development are significant from the energy consumption perspective. Based on these factors it is possible to develop a taxonomy for the computer systems, which can be used as a base for the classification tree.

There are a number of researches devoted to the usage of taxonomies for software engineering covering various perspectives. The Softmake company elaborated a taxonomy based on the requirements, design, specification techniques and coding management [12]. However, this classification is inspired by McConnell's Rapid Development System [13], which addresses issues other than energy efficiency.

Another taxonomy, developed by Watson [14], focused on the Software Development process and related tools. This well-detailed taxonomy, however, is related to the external aspects of software development, such as tools or testing techniques, whilst the focus of the Innometrics project is the influence of the internal aspects of the system on its energy efficiency.

In our previous work [15], a taxonomy and some scenarios for the development of mobile applications were introduced. However, it cannot be properly extended to other relevant areas to be considered—such as cloud computing or embedded systems.

Beloglazov et al. [16] established the comprehensive taxonomy focused on energy efficiency. This work is closely related to our project; however, the reason why the given taxonomy cannot be used as a guideline for scenario elicitation is that the research considers databases and cloud systems only. However, some of the classification principles from this paper can be applied to developing a classification tree.

Ramesh et al. [17] developed a taxonomy for energy management in embedded systems, which focused mostly on the power optimization of existing systems.

In this paper, the primary focus is on three types of computer systems: embedded, cloud, and mobile ones. Thus, for starting, we should unify the taxonomy for these domains and later we enrich it with additional aspects, which are significant for the energy efficiency of the related software products. We used a taxonomy for embedded and intelligent systems by the IDC Company (Dugar et al. [18]). The taxonomy focuses on embedded and intelligent systems and considers cloud and mobile ones as their segments (Figure 1).

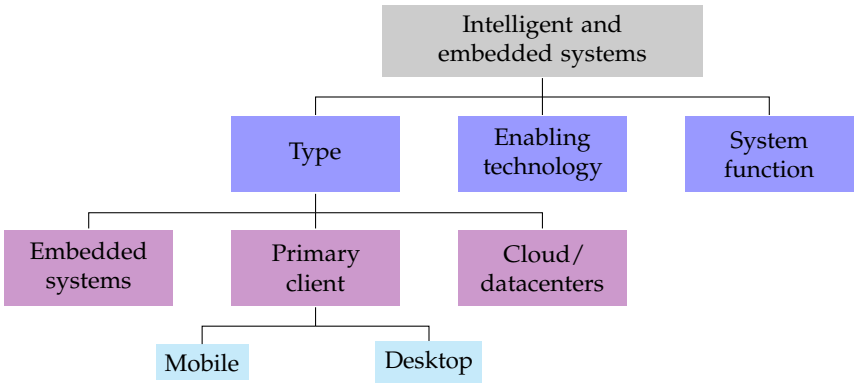



**Figure 1.** Computer systems basic taxonomy based on their type, enabling technology and system function.

The principle of system function classification fits well with the given project. Previously we developed a classification for mobile applications based on the categories from Google Play Store and App Store [15]. In this research that classification will be expanded to cover all types of computer systems. Thus, we combined it with the taxonomy proposed by IDC [18]. The resulting classification (see Table 1) allowed us to classify mobile and desktop applications as well as embedded and cloud systems:

**Table 1.** System function taxonomy for the computer systems.

| System Type | Description |
|---|---|
| transportation | devices and applications for nonautomotive transportation market |
| business | business intelligence systems, digital signage systems, and retail gateways |
| communications | wired and wireless enterprise and service provider infrastructure and client end devices |
| robotics | automotive devices, satellite systems and motion controllers |
| computing | data centers and HPC applications for modeling or data processing |
| healthcare | patient monitoring equipment, diagnostic equipment, and healthcare for healthy people |
| utilities | tools and technologies for controlling and optimizing efficiency of systems or devices |
| entertainment | end-user software like games, media applications |

If we consider the traditional software development lifecycle (in short, SDLC), the taxonomies shown above are related mostly to the project start and early requirements elicitation stages, as far as they describe the type and general purpose of the system.

In a typical development lifecycle, the next stage is the design of the system. At this stage, the application of design patterns may strongly influence the quality of the software product with respect to energy efficiency. The effect of patterns is investigated in a number of research papers [19–21]. Noureddine and RajanIn [21] observed 21 design patterns from an energy efficiency perspective. For the developing taxonomy, we divided these patterns into four categories based on their impact on the energy efficiency of the system: high positive (more than 10% overhead), low positive (less than 10%), low negative (more than −10%), and high negative (less than −40%). The resulted taxonomy is given in Figure 2.

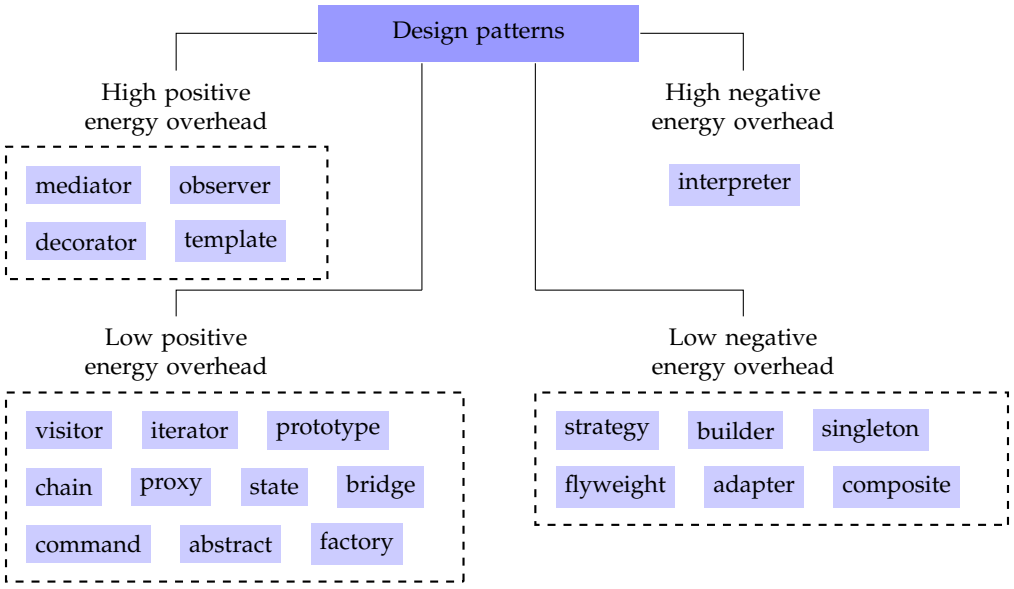



**Figure 2.** Object oriented design patterns taxonomy for the computer systems.

The next stage in the traditional SDLC process is the implementation, i.e., the coding stage. In the previous work [15] we considered development tools (IDE) as one of the aspects that have an impact on energy efficiency. However, we did not find any evidence of this hypothesis in the literature, thus we focused on another issue to be investigated at this stage—the choice of the programming language.

Pereira et al. [22] provided a thorough analysis of different programming languages and their impact on energy consumption. The authors categorized 27 languages based on their paradigm and processing principle (see Figure 3) and analyzed the efficiency of each language, paradigm, and processing principle in performing 10 typical algorithms. However, the categorization based on the language paradigm provides no valuable information regarding the energy efficiency of the programming language (see Figure 4a). The processing principle, on the contrary, allows us to reason about the energy efficiency of the given language, providing a clear distinguishing factor between compiled, interpreted, and virtual machine languages (see Figure 4b).

The compiled languages are the most energy-efficient ones, whilst the group of interpreted languages shows the worst results. Thus, for the developing energy-efficiency taxonomy, the processing principle was selected as the criterion for the classification of programming languages.

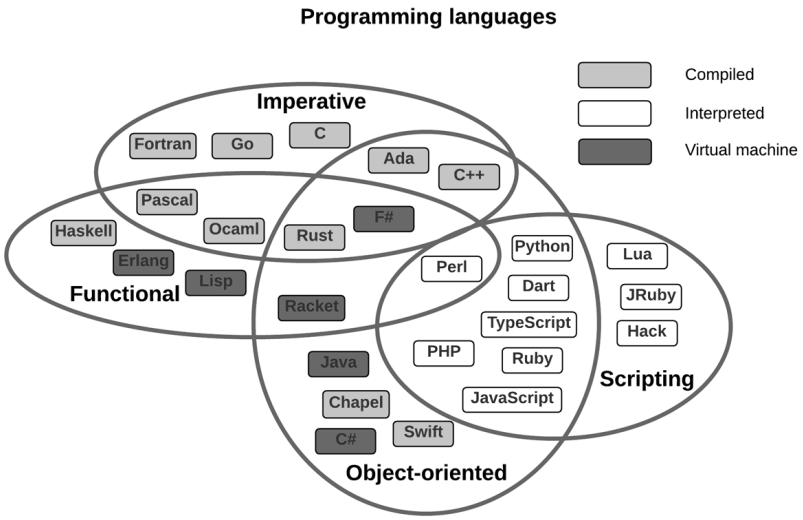
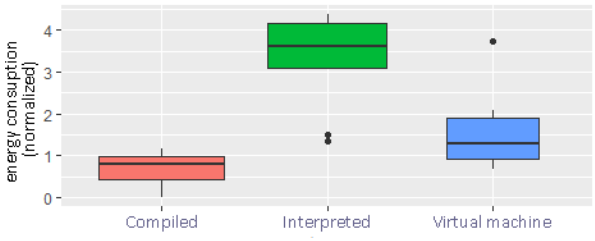
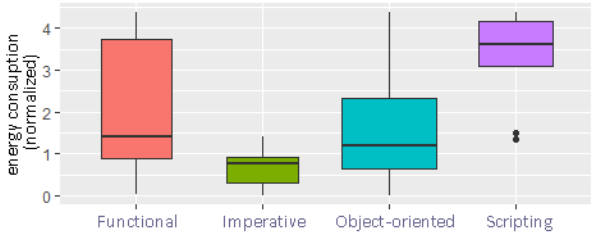



**Figure 3.** Classification of programming languages based on the paradigm and processing principle.




(**a**) Comparison by processing principle.




(**b**) Comparison by language paradigm.

**Figure 4.** Analysis of the energy efficiency of programming languages.

In our previous research [15] we analyzed energy efficiency as a software quality attribute. Based on this, we added the parameter of SDLC to the final taxonomy, as Inukollu et al. [23] argued about the impact of the development lifecycle on the software quality. The taxonomy of SDLC was based on the classification given by Zima [24], therefore we used the same concepts in this research and included traditional, agile, open-source, and individual development as possible options for the SDLC choice.

Another important aspect of energy efficiency analysis is the usage of particular hardware components by the software in operation mode, like sensors or connection channels. It is a very challenging task to classify all possible variants in this area since the number of existing I/O devices is very huge. For the given task we decided to use the taxonomy proposed by IDC [18] and enriched the sensor's section with components analyzed by Javed et al. [25]. The result of the classification is given in Figure 5.

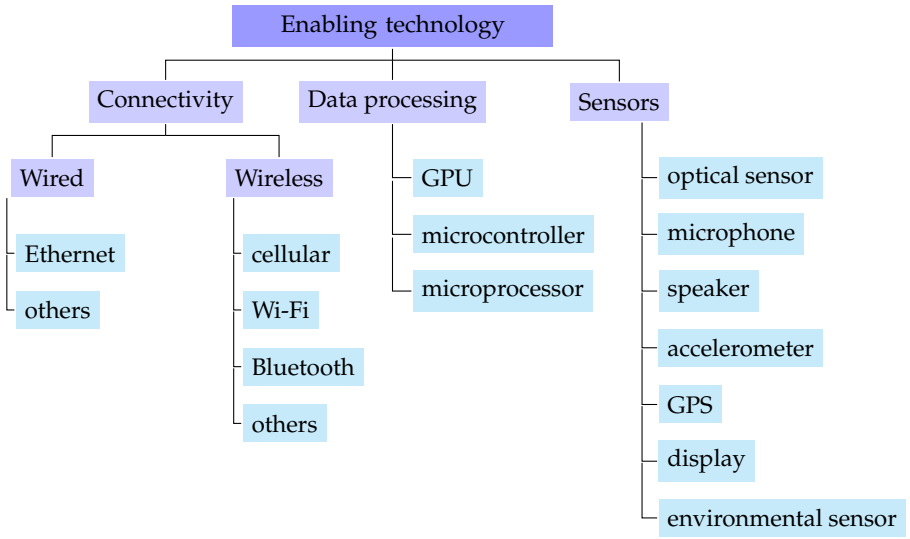



**Figure 5.** Enabling technology taxonomy for computer systems, including connectivity principle, data processing approach and use of sensors.

It should be taken into account that this branch of taxonomy is based on multiple aggregation principles, which means that a system can include several sensors or connections.

The last step in taxonomy development is the expanding of the basic topology given in Figure 1. Since the focus of our research is energy efficiency in mobile, cloud, and embedded systems, we performed decomposition of two categories in the "system type" branch: primary client, which includes mobile systems, and cloud/datacenter system. The former is augmented with our taxonomy for mobile applications [15], and for the latter, some of the criteria from the work by Beloglazov et al. [16] were chosen. We referred to the datacenter level taxonomy, so for this topology type the parameters virtualization, workload, and target systems type are added. Also, we picked out desktop systems from the primary client category and decomposed them based on the operating system type.

The final classification tree is given in Figure 6.

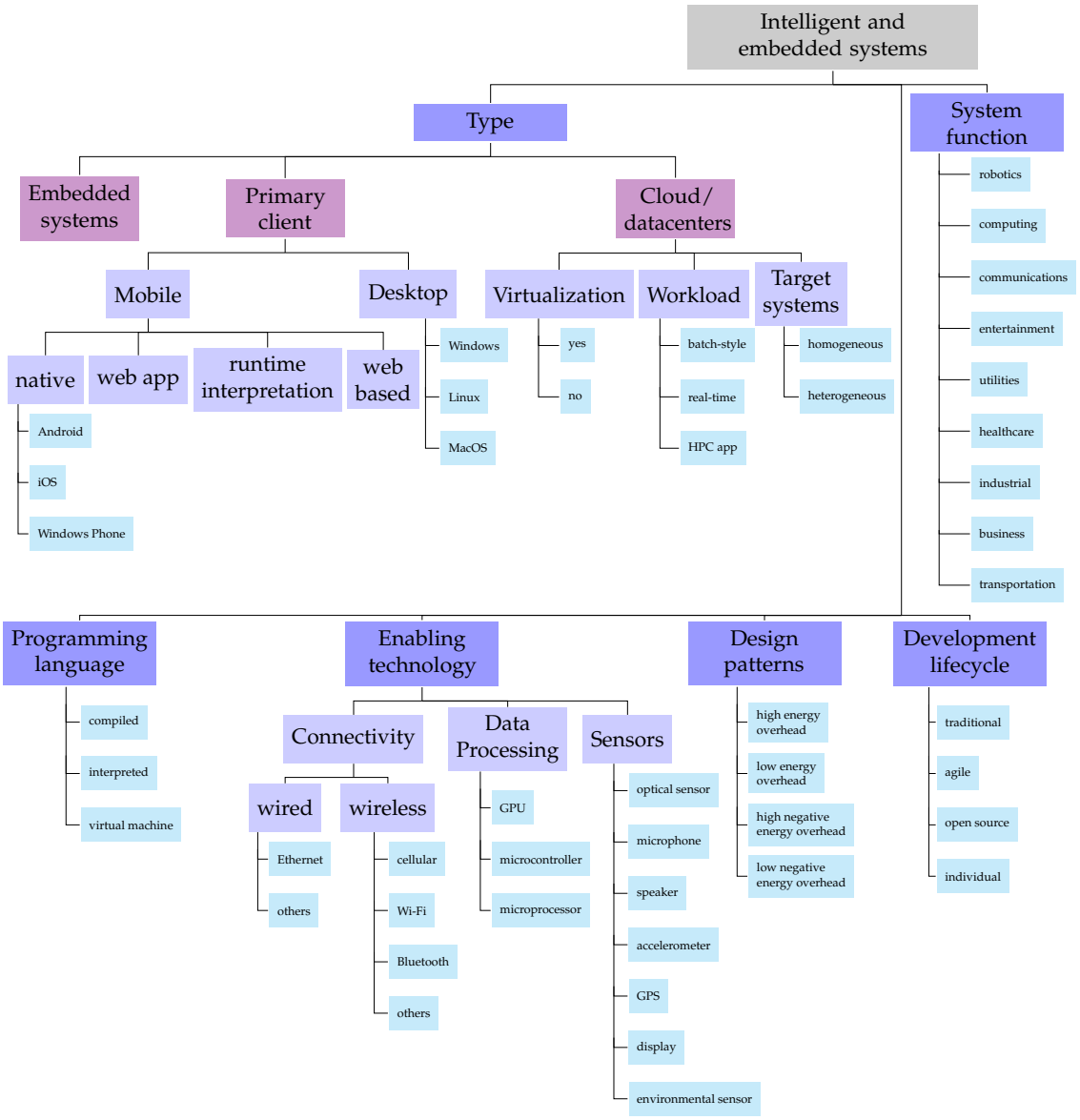



**Figure 6.** Computer system classification tree from the energy efficiency viewpoint.

## 3. Scenario Generation

The combinatorial method used for generating scenarios is obtained by implementing the ACTS (Automated Combinatorial Testing for Software) tool [26] (Figure 7). Firstly, the main classification tree was created including all two-level decompositions. For the categories "System function", "Design patterns", "Programming languages" and "Development lifecycle", the enumerations, describing particular instances or clusters (as shown in Figure 6), were introduced first. For the branch "Enabling technology" we group wired and wireless connectivity options in one sub-category as far as in the taxonomy they are mutually exclusive (which is not true in real life). Also, the sub-category "Sensors" was introduced, as far as ACTS does not support multiple aggregated instances for one parameter. However, we should remember that for this category we are able to choose any subset. The most challenging task was to describe the "Type" category, as far as some of the instances have 2–3 additional levels of decomposition. So, instead of "Mobile" instance, six separate instances were created based on mobile taxonomy. For some systems additional fields were introduced, and for other system types the default value "-" was assigned.

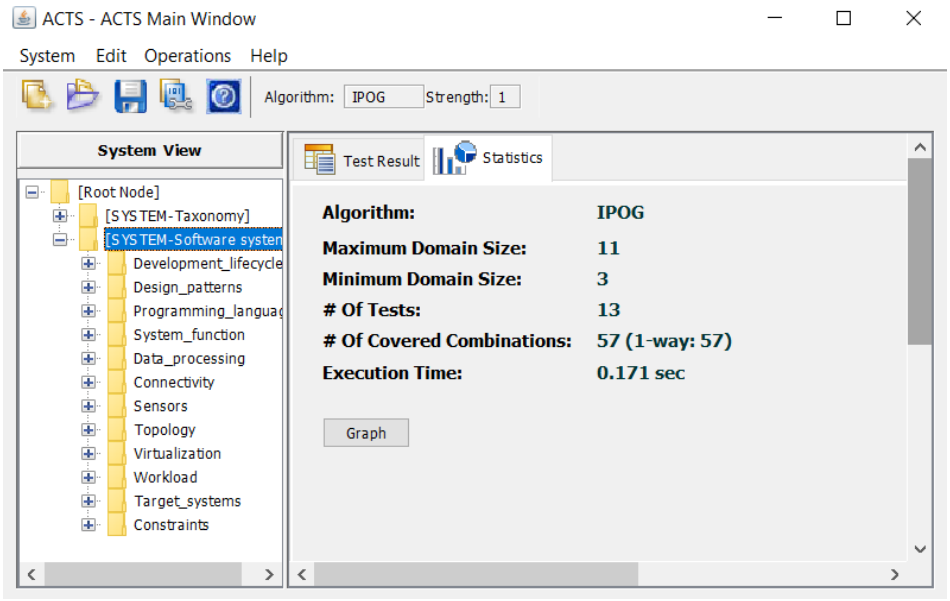



**Figure 7.** Building classification trees and combinatorial testing with Automated Combinatorial Testing for Software tool.

Also, a number of constraints were added to the classification tree before building. The constraints in ACTS syntax are the following:

```
(Topology = "Mobile native Android" || Topology = "Mobile native iOS" ||
Topology = "Mobile native Windows Phone" || Topology = "Mobile web app" ||
Topology = "Mobile runtime interpretation" || Topology = "Mobile web based") =>
(Connectivity = "cellular" || Connectivity = "Wi-Fi")
(Topology = "Embedded") => (Data_processing = "microcontroller")
(Topology != "Embedded") => (Data_processing != "microcontroller")
(Topology != "Mobile native Android" && Topology != "Mobile native iOS" &&
Topology != "Mobile native Windows Phone" && Topology != "Mobile web app" &&
Topology != "Mobile runtime interpretation" && Topology != "Mobile web based") =>
(Connectivity != "cellular")
(Sensors = "GPS") => (System_function = "navigation")
(Topology = "Cloud") => (Connectivity != "Bluetooth")
(Topology = "Desktop Windows" || Topology = "Desktop Linux" ||
Topology = "Desktop MacOS") =>
(Connectivity = "ethernet" || Connectivity = "Wi-Fi")
(Topology = "Cloud") => (Sensors = "no")
(Topology != "Cloud") => (Virtualization = "-")
(Topology != "Cloud") => (Workload = "-")
(Topology != "Cloud") => (Target_systems = "-")
(Topology = "Cloud") => (Virtualization != "-")
(Topology = "Cloud") => (Workload != "-")
(Topology = "Cloud") => (Target_systems != "-")
(Topology = "Mobile native Android" || Topology = "Mobile native iOS" ||
Topology = "Mobile native Windows Phone") =>
(Programming_language = "compiled" ||
Programming_language = "virtual machine")
(Topology = "Mobile runtime interpretation") =>
(Programming_language = "interpreted" ||
```

```
Programming_language = "virtual machine")
(Topology = "Mobile web app" || Topology = "Mobile web based") =>
(Programming_language = "interpreted")
```

For *t*-way test case generation, the IPOG algorithm [27] was used. The variations of one-way, two-way, and three-way testing were observed. The resulted number of test cases is given in Figure 8.

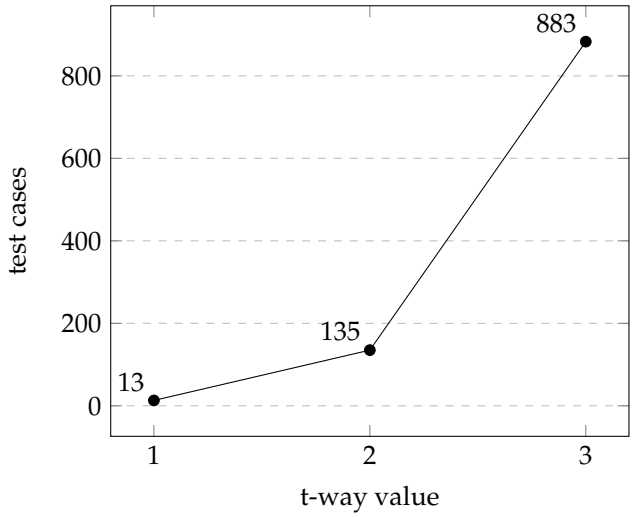



**Figure 8.** The number of generated test cases for various *t*-way options.

As can be seen in Figure 8, the number of scenarios increased exponentially with a higher *t* number. In fact, for the case of scenario development, it is enough to check each instance at least once to conclude about the effectiveness of the developing model. As far as 1-way testing covers all instances, there is no need to provide an extensive investigation based on "all-couples" or "all-triples" combinations.

## 4. Results and Discussion

The resulting test cases for each usage scenario could be summarized as follows (Table 2).

**Table 2.** Test cases for embedded, mobile and desktop systems.

| # | Test Case Description |
|---|---|
| 1 | Server software for HPC application in the homogeneous environment using virtualization. The processing performed primarily on GPU, supporting Wi-Fi connection. The interpreted programming language and high overhead design pattern are used in software implementation. The product is developed by the team according to agile SDLC. The purpose of the software is computations. |
| 2 | Embedded system with optical sensor connected via Bluetooth. The software is open-sourced, written with virtual machine language according to a low non-overhead pattern. The product is developed by the team according to the traditional (waterfall) SDLC. It is purposed for the robotics market. |
| 3 | Mobile native application for Android system with GPU processing and connection via cellular network 3G/4G module. It is coded with compiled language according to the high non-overhead pattern by an individual person. The purpose of the application is entertainment area, thus usage of such sensors as optical sensors and display could be assigned to this scenario. |

**Table 2.** *Cont.*

| # | Test Case Description |
|---|---|
| 4 | Mobile native application for the iOS system based on CPU computations and connected via Wi-Fi. Software is written in a compiled language based on a low overhead design pattern by a development team with agile SDLC. Its purpose is communications. |
| 5 | Mobile native application for Windows Phone. In operation mode, it uses a Wi-Fi connection and GPU to perform computations. It is written with virtual machine language according to a high non-overhead pattern by an individual developer. It is positioned as a healthcare product. Based on the software function area, the possible option is the usage of accelerometer and GPS |
| 6 | Desktop Windows-based software. It uses an Ethernet connection and performs computations in GPU. The software is written with compiled language according to a low non-overhead pattern by a development team based on traditional SDLC. The software is developed for an industrial system, so a number of optical and environmental sensors, as well as display, are assigned to this scenario |
| 7 | Desktop software under Linux OS. In operation mode, it uses an Ethernet connection and performs computations in the CPU. The software is written with interpreted language using a high overhead design pattern. The software is developed by an individual person. It is targeted to the business systems, so no special sensors are used in this case, except display |
| 8 | Desktop software for macOS. In operation mode, it uses a Wi-Fi connection and performs computations in GPU. The software is written with interpreted language using a high non-overhead design pattern. This is an open-source project. It is targeted to the utilities market |
| 9 | Mobile web application with CPU processing and connection via a cellular network. It is written in the interpreted language according to a high overhead design pattern. It is an open-source project, targeted to the entertainment sector. It was decided to assign the speaker and microphone sensors to this case to specify the software purpose and operation pattern |
| 10 | Mobile application with runtime interpretation. It operates under Wi-Fi connection with CPU processing. It is written with interpreted language and a high non-overhead design pattern. The software is developed by the team with a traditional SDLC. The target market is healthcare. The external environmental sensors are used with this software as the source of input data |
| 11 | Mobile web-based application using a Wi-Fi connection and CPU in normal operation mode. The software is written by an individual developer with interpreted language following a low non-overhead design pattern. It is dedicated to the transportation sector. No specific sensors assigned to this scenario |
| 12 | Cloud-based software for real-time processing in heterogeneous systems. Computations performed on CPU, connection type is not specified. The software is developed with virtual machine language based on a high non-overhead design pattern by an agile development team. It is dedicated to the utilities sector |
| 13 | Cloud-based solution for batch-style workload designed for homogeneous systems. It implies virtual machine, computations performed on CPU, a connection is made via Ethernet. The software is developed with interpreted language based on a high overhead design pattern. It is an open-source project, targeted to the entertainment market |

This is the data obtained by using combinatorial testing for the developed taxonomy in the form of a classification tree. However, it is not a proper usage scenario definition, as far as these cases are not sufficiently detailed. The last step to define scenarios is to specify each generated test case and,

whenever is possible, provide a link to the real projects so that they can be used as valid cases in further research.

This part is performed based on the empirical analysis and expert judgment of the research group. The main criteria for scenario definition were the existence and accessibility of the particular software development projects for the thorough analysis from an energy efficiency perspective. In particular, for the University of Bologna, the case studies will refer to embedded systems related to the military domain. For Innopolis University, the case studies will start within the research labs, and be performed together with partner companies located in the Innopolis; later they will be extended to the major Russian software producers.

The list of the relevant case studies is given below.

Case study 1. The scenario refers to a scientific application, using Python as programming language. In this context, the possible case study is an R&D project of Innopolis University devoted to the development of the geodesically accurate digital model of the territory of the Republic of Tatarstan [28].

Case study 2. This context correlates with the Smart TV app, as far as Java is used in Android TV development and usage of Bluetooth connection can be used in control signal transmitting. The scenario is inspired by the case study of Sitronics Telecom Solutions project for Smart TV systems.

Case study 3. Android-based application written on Java using Android Studio. This scenario fits well with the game development project. The game should include real-world interaction or augmented reality as far as active usage of GPS is assumed. The examples of such projects are Pokemon Go or Geocaching [29].

Case study 4. Mobile software development project for the iOS system. For this scenario, the case study of ABDT company's project of mobile bank application can be assigned. The iOS version of the app is developed on Swift and Objective-C using XCode IDE. There are several teams working on the project using an agile approach based on SAFe methodology [30].

Case study 5. Windows Phone app developed on C# in the Visual Studio. The app represents a fitness tracker. As far as it is an individual development and not a large-scale project it is possible to bring this scenario to life as the student's course project.

Case study 6. The development of the control system based on machine vision. The software is developed in C++ using Microsoft Visual Studio involving CUDA for GPU computations. The traditional SDLC is usually used in government-funded projects. Thus, the case study could be derived from the analysis of open databases of such funds as Fund for Assistance to Small Innovative Enterprises (FASIE), Skolkovo, or international foundations. The example of a system that operates on Windows OS as well as matches the requirement of GPU computations and the usage of optical sensors is the recently launched project "Monitoring and quality control system for iron ore raw materials processing" [15,31] supported by FASIE.

Case study 7. The PC software for the Linux system is written in Python. No additional requirements are put on this scenario. Thus, a student's course project could be specified to get a relevant case study.

Case study 8. The scenario describes the development of a framework for modeling and simulation of some physical processes or aggregates. It is implemented in Objective-C. The possible case for this scenario is the open-source project SOFA—an efficient framework dedicated to research, prototyping, and development of physics-based simulations [32].

Case study 9. The scenario of a simple mobile audio player implemented on JavaScript in JQuery Mobile framework. It can be the open-source project, such as MediaElement.js [33]

Case study 10. JavaScript app developed in React Native or Xamarine IDEs. Based on the application area and specifics of external environmental sensors, the app is a kind of a medical assistant, such as blood pressure measurement, eye care, or diabetes journal apps [34].

Case study 11. Mobile app built with JavaScript on Adobe PhoneGap framework. A number of possible case studies can be assigned to this scenario. As far as individual development is required, the student's course work project could be elaborated as a relevant case study.

Case study 12. The possible scenario implementation is a Java project. The description of the test case together with the given specifics ideally matches the CIRI ICT project of the Università di Bologna dedicated to the resource management platform for cloud computing applications [35].

Case study 13. The scenario is related to the cloud-based open-source platform. It should be based on the Perl language. The examples of such open-source projects are WebGUI CMS [36] or Movable Type publishing platform [37].

## 5. Conclusions

Within the scope of this study, a thorough analysis of various taxonomies of different kinds of computer systems with respect to energy consumption was performed. We developed a taxonomy focused on the properties of computer systems that have an impact on energy efficiency.

The obtained set of scenarios based on a combinatorial analysis of embedded and intelligent systems' taxonomy provides full coverage of possible case studies for the tools and models developed within the project of energy efficiency analysis of the software under varying technological contexts (e.g., cloud, mobile, embedded). It has to be mentioned that we can use various optimization techniques for the purpose of creating the reliable set of validation scenarios from the derived classification tree. Approaches such as Monarch Butterfly Optimization (MBO) [38], EarthWorm optimization Algorithm (EWA) [39], Elephant Herding Optimization (EHO) [40], Moth Search (MS) algorithm [41] and others could be used to generate compact validation sets. However, the focus of the paper is not to analyze different combinatorial algorithms, but to show the general idea of using them to generate a set of validation scenarios.

The defined scenarios will be used in the empirical experimentation part of the project to validate the hypothesis that our methods and tools are able to address practical needs. From the given scenarios the feedback to the researchers and developers of the current research project will be obtained. Such feedback will be used to improve the tools and the way they are integrated.

The derived set of scenarios was augmented with a specific context and additional information about particular tools, methods, and approaches used in the development process. Finally, we come up with a set of case studies related to the context of a particular scenario. Case studies will be shared by the research unit working on the same scenarios thus allowing the comparison of the results and the identification of the peculiarities of the different techniques.

The primary goal of the research was to develop a set of scenarios to define relevant case studies for further industrial testing of the developed energy efficiency assessment framework. However, we hope that the given taxonomy will be useful for the researches in computer system categorization and classification.

**Author Contributions:** Conceptualization, P.C. and G.S.; methodology, A.K.; validation, P.C., A.S. and E.Z.; formal analysis, A.K.; investigation, A.K., G.S. and A.S.; resources, P.C. and G.S.; data curation, E.Z.; writing—original draft preparation, A.K.; writing—review and editing, A.S.; supervision, G.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research project is carried out under the support of the Russian Science Foundation Grant No. 19-19-00623.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. Corral, L.; Georgiev, A.B.; Sillitti, A.; Succi, G.; Vachkov, T. Analysis of Offloading as an Approach for Energy-Aware Applications on Android OS: A Case Study on Image Processing. In Proceedings of the International Conference on Mobile Web and Information Systems, Barcelona, Spain, 27–29 August 2014; pp. 29–40. [CrossRef]
2. Fiksel, J. A framework for sustainable materials management. *JOM* **2006**, *58*, 15–22. [CrossRef]
3. Ronchetti, M.; Succi, G.; Pedrycz, W.; Russo, B. Early estimation of software size in object-oriented environments a case study in a CMM level 3 software firm. *Inf. Sci.* **2006**, *176*, 475–489. [CrossRef]
4. Weiser, M.; Welch, B.; Demers, A.; Shenker, S. Scheduling for reduced CPU energy. In *Mobile Computing*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 449–471.
5. Bekas, C.; Curioni, A. A new energy aware performance metric. *Comput. Sci. Res. Dev.* **2010**, *25*, 187–195. [CrossRef]
6. Ciancarini, P.; Ergasheva, S.; Kholmatova, Z.; Kruglov, A.; Succi, G.; Vasquez, X.; Zuev, E. Analysis of Energy Consumption of Software Development Process Entities. *Electronics* **2020**, *9*, 1678. [CrossRef]
7. Ambler, S.W. Usage Scenarios: An Agile Introduction. Available online: http://agilemodeling.com/artifacts/usageScenario.htm (accessed on 20 October 2020).
8. de Oliveira, A.D.; Filomena, T.P.; Righi, M.B. Performance comparison of scenario-generation methods applied to a stochastic optimization asset-liability management model. *Pesqui. Oper.* **2018**, *38*, 53–72. [CrossRef]
9. Heugens, P.P.; van Oosterhout, J. To boldly go where no man has gone before: Integrating cognitive and physical features in scenario studies. *Futures* **2001**, *33*, 861–872. [CrossRef]
10. Van Notten, P. Scenario Development: A Typology of Approaches. 2006. Available online: https://www.oecd.org/site/schoolingfortomorrowknowledgebase/futuresthinking/scenarios/37246431.pdf (accessed on 9 December 2020).
11. Bruninx, K. Improved Modeling of Unit Commitment Decisions under Uncertainty. Ph.D. Thesis, KU Leuven Faculty of Engineering Science, Leuven, Belgium, 2016.
12. Softmake's Software Development Taxonomy. Available online: https://www.softmake.com.au/softwareDevelopmentMethodology/softmakesSoftwareDevelopmentTaxonomy.html (accessed on 10 July 2019).
13. McConnell, S. *Rapid Development: Taming Wild Software Schedules*, 1st ed.; Microsoft Press: Redmond, WA, USA, 1996.
14. Watson, C. Software Development Tools Taxonomy. Available online: https://craigwatson1962.wordpress.com/2010/12/19/software-development-tools-taxonomy/ (accessed on 10 July 2019).
15. Ivanov, V.; Kruglov, A.; Sadovykh, A.; Succi, G. Scenarios for the evaluation of the energy efficiency of mobile applications. In Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 17–19 October 2019.
16. Beloglazov, A.; Buyya, R.; Lee, Y.C.; Zomaya, A. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 2011; pp. 47–111. [CrossRef]
17. Ramesh, U.B.K.; Sentilles, S.; Crnkovic, I. Energy management in embedded systems: Towards a taxonomy. In Proceedings of the 2012 First International Workshop on Green and Sustainable Software (GREENS), Zurich, Switzerland, 3 June 2012, [CrossRef]
18. Dugar, A.; Rau, S.; Turner, N.; Palma, M.J.; Santiago, L. IDC's Worldwide Embedded and Intelligent Systems Taxonomy, 2017: Views by Internet Topology, System Function, and Enabling Technology. Available online: https://docplayer.net/63297958-Idc-s-worldwide-embedded-and-intelligent-systems-taxonomy-2017-views-by-internet-topology-system-function-and-enabling-technology.html (accessed on 12 July 2019).
19. Feitosa, D.; Alders, R.; Ampatzoglou, A.; Avgeriou, P.; Nakagawa, E.Y. Investigating the effect of design patterns on energy consumption. *J. Softw. Evol. Process.* **2017**, *29*, e1851. [CrossRef]
20. Litke, A.; Zotos, K.; Chatzigeorgiou, A.; Stephanides, G. Energy Consumption Analysis of Design Patterns. In Proceedings of the International Conference on Machine Learning and Software Engineering, Porto, Portugal, 3–7 October 2005; pp. 86–90.

21. Noureddine, A.; Rajan, A. Optimising Energy Consumption of Design Patterns. In Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16–24 May 2015. [CrossRef]
22. Pereira, R.; Couto, M.; Ribeiro, F.; Rua, R.; Cunha, J.; Fernandes, J.P.; Saraiva, J. Energy efficiency across programming languages: How do energy, time, and memory relate? In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, Vancouver, BC, Canada, 23–24 October 2017, [CrossRef]
23. Inukollu, V.N.; Keshamon, D.D.; Kang, T.; Inukollu, M. Factors Influencing Quality of Mobile Apps: Role of Mobile App Development Life Cycle. *Int. J. Softw. Eng. Appl.* **2014**, *5*, 15–34. [CrossRef]
24. Zima, D. Modern Methods of Software Development. *TASK Q.* **2015**, *19*, 481–493.
25. Javed, A.; Shahid, M.A.; Sharif, M.; Yasmin, M. Energy Consumption in Mobile Phones. *Int. J. Comput. Netw. Inf. Secur.* **2017**, *9*, 18–28. [CrossRef]
26. Kuhn, R. Automated Combinatorial Testing for Software. Available online: https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software (accessed on 19 July 2019).
27. Nie, C.; Leung, H. A survey of combinatorial testing. *ACM Comput. Surv.* **2011**, *43*, 1–29. [CrossRef]
28. Digital Model of Tatarstan Republic. Available online: https://docplayer.ru/43169348-Proekt-cifrovaya-model-respubliki-tatarstan.html (accessed on 21 September 2019).
29. Best GPS Location-Based Games on iOS and Android 2018. Available online: https://www.redbytes.in/gps-mobile-game-development-ios-android-2018/ (accessed on 21 September 2019).
30. Scaled Agile Framework. Available online: https://www.scaledagileframework.com/ (accessed on 21 September 2019).
31. Kruglov, V.N. Using Open Source Libraries in the Development of Control Systems Based on Machine Vision. In *IFIP Advances in Information and Communication Technology*; Springer International Publishing: Midtown Manhattan, NY, USA, 2020; pp. 70–77. [CrossRef]
32. Simulation Open Framework Architecture. Available online: https://www.sofa-framework.org/ (accessed on 21 September 2019).
33. MediaElement.js. Available online: https://github.com/mediaelement/mediaelement (accessed on 21 September 2019).
34. Medical Applications. Available online: https://mobile.softpedia.com/windows-phone/medical (accessed on 21 September 2019).
35. CIRI Information and Communication Technologies Research Projects. Available online: http://www.ciri-ict.unibo.it/en/research (accessed on 21 September 2019).
36. WebGUI Content Engine. Available online: http://www.webgui.org/ (accessed on 21 September 2019).
37. MovableType Project. Available online: https://movabletype.org/ (accessed on 21 September 2019).
38. Wang, G.G.; Deb, S.; Cui, Z. Monarch butterfly optimization. *Neural Comput. Appl.* **2019**, *31*, 1995–2014. [CrossRef]
39. Wang, G.G.; Deb, S.; Coelho, L.D.S. Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems. *Int. J. Bio-Inspired Comput.* **2018**, *12*, 1–22. [CrossRef]
40. Li, J.; Lei, H.; Alavi, A.H.; Wang, G.G. Elephant Herding Optimization: Variants, Hybrids, and Applications. *Mathematics* **2020**, *8*, 1415. [CrossRef]
41. Wang, G.G. Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Comput.* **2018**, *10*, 151–164. [CrossRef]