InnoMetrics Dashboard: The Design, and Implementation of the Adaptable Dashboard for Energy-Efficient Applications Using Open Source Tools

Shokhista Ergasheva, Vladimir Ivanov, Ilya Khomyakov, Artem Kruglov, Dragos Strugar, Giancarlo Succi

InnoMetrics Dashboard: The design, and implementation of the Adaptable Dashboard for Energy-Efficient Applications using Open Source tools

Shokhista Ergasheva[1], Vladimir Ivanov[1], Ilya Khomyakov[1], Artem Kruglov[1], Dragos Strugar[1], and Giancarlo Succi[1]

Innopolis University, Russian Federaton, 420500, Tatarstan Republic
s.ergasheva@innopolis.university,
{v.ivanovov, i.khomyakov, a.kruglov, d.strugar, g.succi}@innopolis.ru

Abstract. Increasing amount of data the organizations worldwide have at their disposal lead to the need to structure, organize and present the information obtained from it. That is because, in today's rapid-changing business environment, managers and executives need to be able to gain crucial insights about the ongoing project in as little time as possible. Recently, energy efficiency has become a greater field of research, and companies started concentrating on monitoring energy-related metrics. In addition, many of them have built their own internal tools (dashboards) to do just this. However, one of the major drawbacks of building specialized tools is the lack of adaptability. That is, they are often tailored to only one person (e.g. CEO), or a small group of them (e.g. board of directors, managers). Furthermore, the combination of metrics that is displayed to them does not change over time. This is a problem because most likely there exists a better metric combination that would allow users to get the crucial insights faster. To fill this gap, our ongoing research focuses on making the dashboards adaptable to multiple roles within the organization while optimizing for a certain goal. In some scenarios the dashboard's goal may be to detect defects, in others it may be to generate the most profit. As our primary research interest is to amplify energy efficiency, we have chosen that to be our dashboard's goal. Our previous work suggests that in order to handle compound metrics at scale it is needed to represent the dashboard as a complex system. This paper presents the design and the architecture of our proposed solution synergizing the notions from complexity theory, software architecture and user experience (UX) design.

Keywords: Dashboards · Energy Efficiency · Adaptable Systems

1 Introduction

Due to modern-day rapid-changing business requirements, especially in the mobile space, and the advent of agile methodologies for tracking project status came

need to continuously monitor the overall picture of energy consumption. At the beginning, it was assumed that only hardware metrics influence energy efficiency levels [1]. Lately, more focus has been given to software (code) and process related metrics [2, 3]. In order to achieve the products' continuous delivery it is necessary in each step of software development process to help the development team to assess energy efficiency levels and suggest ways of potentially improving it. Therefore, our research tries to pinpoint the software, process and project-related metrics that have the influence on energy efficiency, and then optimizing these metrics. This paper highlights the steps we took to design, architect and develop a dashboard specifically made for the use case of monitoring energy efficiency and optimizing the set of parameters needed to be displayed to different users.

1.1 Outline of the paper

We start by mentioning the importance of the problem we are trying to solve as well as the research hypothesis by giving a literature review in subsection 1.2. The primary sections containing the majority of paper's novel contributions are sections 2, 3 and 4 where we decomposed our action plan and outlined all the metrics we used, and major design and architecture decisions. Here we talk at length about the aspects like technology choice, design style guide, layout, widgets, charts, security, complex systems and more. As the solution we have developed is still in its infancy, section 5 pinpoints the key ways our research is going to progress with the dashboard being one of the central tools. Finally, section 6 brings out some reflections.

1.2 Related Work

The field of effective dashboard design has gained traction at the beginning of the new millennium thanks to Stephen Few, among others. His "Information dashboard desig" [4] is still being referenced by many. He starts by arguing that dashboards in the 1990s had major flaws, and that their root problem was in the inability to effectively communicate. Few also said that dashboards have to hold a balance between clarity and complexity [5]. With recent advances in the field of User Experience (UX) and User Interface (UI) design, dashboards started getting significantly more traction [6–10]. This resulted in dashboards being way more able to focus on the communication aspect, thus increasing use cases, leading to their wide-spread adoption [4]. Dashboard are becoming also of stragegic importance in lean and agile environments [11, 12], to spread knowledge across workers [13, 14], and in distributed development environments [15–17].

Meanwhile, with the increasing popularity of mobile devices, energy efficiency has become an ever-growing concern. Recent research by Pinto et al. [2] pinpointed that energy efficiency has to be addressed at all levels of the software stack. More specifically, they argue that it is due to the developers having the

lack of knowledge and lack of tools. These bottlenecks prevent software developers from identifying, fixing, refactoring, and removing energy consumption hotspots [18].

Finally, it would be advisable to take an open source approach to this topic, since it would ensure a larger diffusion of the dashboard [19–21].

2 Energy-Efficiency related metrics

In this section we describe the importance of metrics included in the dashboard we are developing. We start by providing an overview of the project as well as additional information to place the system in context. Firstly, the dashboard we are building is mainly concerned with monitoring the energy efficiency of software artifacts. The goal is to make it easy and convenient for managers as well as other users to track the energy consumption of the software. Overall, energy consumption metrics referring to the energy consumption can be divided into:

- Process metrics
- Project metrics
- Product metrics

We provide several Process and Product metrics throughout this paper that encapsulate the functionality of the dashboard.

Dashboard's primary purpose is to share important information in terms of different metrics to the end user. Determining what what it is that is *important* to the user is the most critical part of designing the dashboard. By considering the metrics provided one would gain crucial insights, allowing them to make more educated decisions faster.

By working closely with the industry representatives, as well as consulting the literature in this field (data and findings revealed in [22–26]), we have come up with the list of metrics that the potential users would find most valuable [27–29] (Table 1).

These metrics are of paramount importance in qualifying the results of the work so that developers can judge objectively the development status. The main contribution can be directed into the energy efficient development of software in teams and individually. The proposed metrics were computed by applying rules common to the existing hierarchical measures of other internal software attributes. For example, by knowing the Number of classes in the source code we can easily calculate other metrics like: Number of Immediate sub-classes of a Class, Response for a Class, Weighted methods per Class as well as the Number of Children metric. It can be followed by better development estimates and more qualified product that can maintain the customers expectations. As it was stated in the paper [25], developers need a set of valid indicators in assisting the developers for better applications. Thus, the study results showed that Cyclomatic complexity, Number of classes, LOC, number of parameters, nested block depth, number of methods, weighted methods per class and method

lines have direct relationship with power consumption of mobile applications. Authors of the paper [22] demonstrated specific coding practices for reading array length information, accessing class fields and performing invocations help to lower the overall energy consumption and improve the usability of the software.

Expanding the list of energy consumption metrics is already on our research agenda. However, not only obtaining additional metrics is of high significance. Combining the already existing ones, as well as easily modifying them is very important. Thus, section 4 describes how we utilized these metrics in greater detail.

Table 1. Code metrics for the energy efficiency measurement

#	Name of the metric	Description
1	Performance tips	Strategies that can be used to conserve and extend the battery life.
2	Code size	The length of the code and row column weight in the form.
3	Loops	Number of For and While loops in the source code.
4	Code Smells	Internal Setter, Leaking Thread, Member ignoring method and Slow Loop consume up to 87 times more.
5	Cyclomatic complexity	It counts the number of flows through a piece of code.
6	Classes	The number of classes used in the source code.
7	LOC (XML)	Lines of code in XML.
8	LOC (Java)	Lines of code in Java.
9	Collections types in Java	Number of frameworks that provide an architecture to store and manipulate the group of objects in Java.
10	Invocation	Number of invocations of a program or a function.

3 Design, Visualization & Development

3.1 Design

S. Few in his [30, 4, 5, 31] pointed out several key characteristics that all dashboard should have. This section makes a bridge between the section 2 and the design guidelines. As he (Few) puts it,

"A dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance."

We strongly agree with the majority of his guidelines. First, all elements of a dashboard have to have a purpose and only the most important information that truly contributes to a specified goal should be shown to the end user. A dashboard is a mean of communication, and in order to effectively convey the message to the user, the message has to be clear, concise and goal-oriented.

Message has to be clear The information presented on a dashboard is most commonly a combination of graphics and text, with the emphasis on visual representations. The reason is that visuals can communicate with richer meaning than text on its own. Having clear, easy to understand graphics is one of the crucial requirements from the design perspective.

Message has to be concise It is highly advisable to fit the main metrics of the dashboard to one screen and one screen only, without horizontal and vertical scrolling. The users should be able to easily scan through all the visuals effortlessly and obtain the insights they are looking for. If the system consists of dozens and even hundreds of smaller metrics and they all need to be shown, the ones that are less important should be moved elsewhere. Only the metrics with high level of importance should appear on the main page.

Message has to be goal-oriented Whether it is to gain an important business insight, or to successfully finish a project on time, dashboard's goal is to get users as close to their objectives as possible. What we have found is that a feedback loop, where users would report which metrics are more relevant to the goal than others, is one of the most valuable insights we could get to improve the set of metrics that get shown. This approach will be thoroughly examined later.

3.2 Visualization

As mentioned in the previous subsection, the visual representation and layout play a crucial role in designing a goal-oriented dashboard. Hence, visuals like widgets [32], graphs, charts all require special attention. Thus, we needed to develop a style guide, a set of standards and styles that all of our visual elements would conform to. It would enforce style to improve communication via consistency and clarity. The fully developed style guide can be viewed at: https://innometrics-12856.firebaseapp.com//dashboardhttps://innometrics-12856.firebaseapp.com/.

Here is a comprehensive list of visual elements and containers chosen for the style guide:

- Colors
- Typography
- Container Elements (cards, carousels, modals, tables, etc.)
- Functional Elements (forms, menus, etc.)
- Charts

We have divided colors into three parts: theme colors, grays, and additional colors. This way the theme could be changed, and the users could tune the settings to their liking. Next, typography for headings one through six, as well as **bolded**, *italicized* and underlined text has also been defined. In addition, cross-functional and organizational elements such as forms (with buttons and input fields), tables (with cells), and different types of containers also needed a well-defined set of styling rules.

However, the main page of the dashboard consists almost entirely of widgets and charts. As we noted earlier, it is crucial to keep the main page free of clutter and not have to scroll or click elsewhere to get to the crucial information. That is why, due to their effectiveness and expressive power, we chose to fine tune their visual appearance. Taking multiple existing charting libraries into consideration and comparing their advantages and disadvantages, we decided to

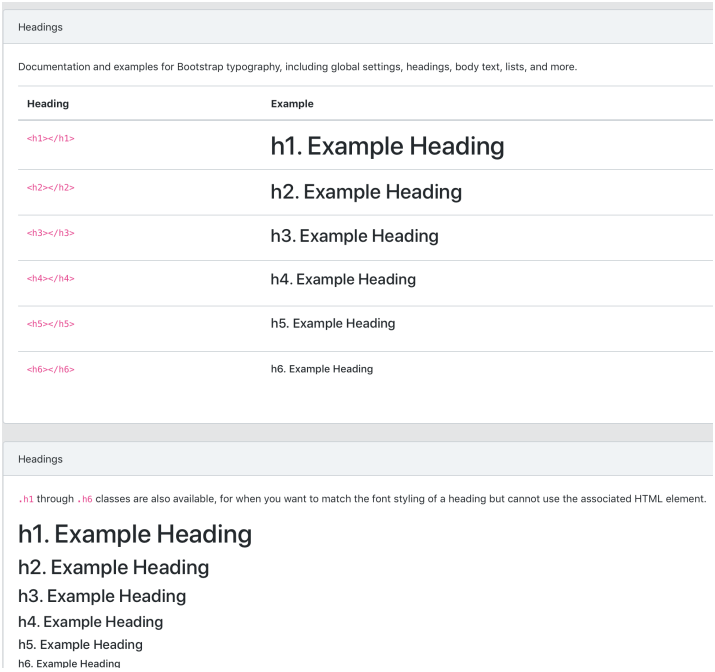

Fig. 1. Typography - Headings 1 through 6

go with https://www.chartjs.org/*ChartJS*. It has all the visuals we wanted, such as bar, line, and area charts and allows for editing when needed. Figure 2 depicts some of them that we decided to include in our prototype.

3.3 Technology

Considering the architectural decisions from section 2 and design decisions from sections 3.1 and 3.2, choosing the platform was relatively straightforward, it had to be a web application. That means that the main programming language for development should be *JavaScript (JS)*.

However, choosing the appropriate JS framework/library was a challenging task. There are numerous options, and all of them offer similar functionality that we need. The one that stood out and ultimately we ended up choosing is *ReactJS*. The primary reason being that its Virtual DOM (Document Object Model) is optimized for real-time applications and that it facilitates the overall process of writing components. Our system could potentially consist of hundreds of metrics, and reusing existing components while at the same time having impressive performance is critical.

In addition, file structure in *ReactJS*, although not opinionated, could be organized in very practical and clear ways. Consider, for example, our code structure, shown in the figure 3. It captures almost all the functionality while

Fig. 2. Different types of Charts from ChartJS

still remaining organized. Notice that we have left some of the implementation parts due to the lack of space.

To manage application's state we use https://redux.js.org/*Redux*, a flexible, and predictable state container. It allows us to easily handle user input, authentication and authorization, as well as storing the results from REST API calls. One of the main benefits of using a state container is the centralization, where application's state and logic enable capabilities like undo/redo, state persistence and much more.

definition

4 Dashboards as Complex Systems

As mentioned in the subsection 3.1, all dashboards must be optimized for a certain objective. As energy efficiency became a greater problem in recent years [33], our research agenda encapsulated that. It was immediately noticeable that the majority of state-of-the-art energy efficient software solutions had a very limited and strictly defined set of metrics to optimize for.

We took a different approach to addressing energy efficiency concerns. Instead of having a fully deterministic system, one in which each and every metric is manually chosen to be displayed, we opted for a system where an algorithm would decide whether or not to display a certain metric. The system would still

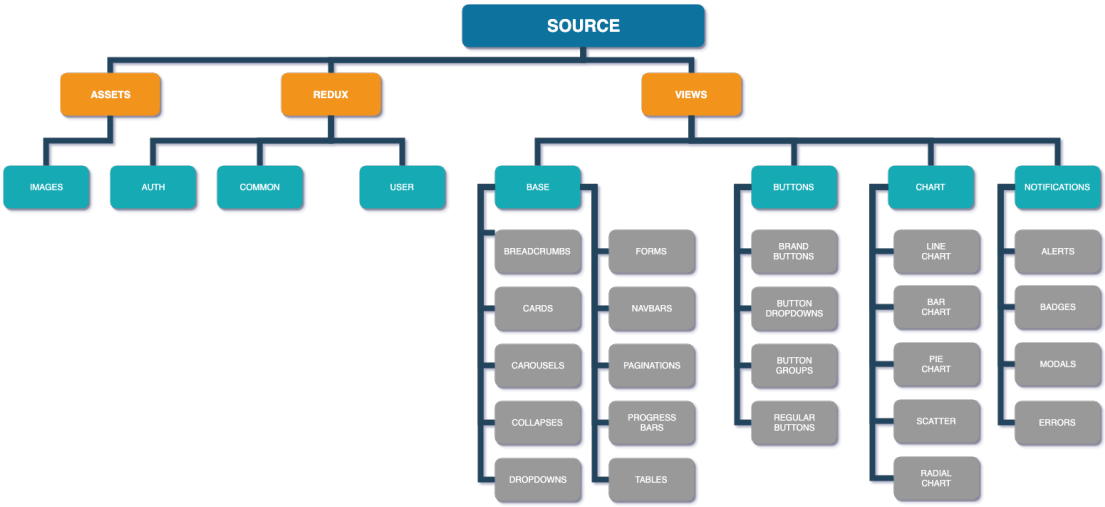

Fig. 3. File Structure of the Implementation

take into consideration the manually entered metrics, albeit not relying on them heavily. All of this would be supported by a feedback loop, where users would rate the importance of the metrics they have been shown.

4.1 Self-adaptable Systems and Energy-Efficiency Dashboards

Self-adaptable systems (systems with behavior just mentioned) have recently gained traction [34], and they could be used as an abstraction to represent the software artifact, such as a dashboard. The rationale behind choosing Complex Systems (CS) to perform the self-adaptation is the following; by definition, a CS is a system with components particularly difficult to model. As it has not yet been fully investigated which factors (metrics) influence the efficiency the most, one approach would be to try and make all combinations of metrics and record the results. However, one may notice that as the number of metrics significantly increases, generating all combinations becomes unfeasible. In addition, metrics do not necessarily need to interact with others in linear fashion [35]. In fact, most of the interactions in CS are nonlinear, and often stochastic.

Therefore, we argue that dashboards for energy efficiency applications should be represented as a Complex System. Metrics (presented in section 2), key components of dashboards, become agents within the system. That allows us to closely monitor the relationships each metric has with the other. It became apparent that almost no metrics are isolated, i.e. energy saved depends on a vast majority of software metrics that interact with each other. Thus, carefully monitoring and reacting to change is critical.

By embracing feedback loop and adaptation, these metrics are able to synchronize their internal states with the other metrics in the system. Additionally,

the system should be able to recognize these changes and self-adjust with the emergence of globally coherent patterns of adjustment developing. This way the system would notice a change, and adapt to it.

So far we have only discussed the interactions between the metrics contributing to the global state. However, a major part of any CS is the ability of the system to feed back the globally coherent patterns to micro-level agents. These patterns are discovered either by the system or the dashboard user. An example of a system-discovered pattern in the energy efficiency context would be battery drainage per minute. The system, if noticed irregularities, would notify the agents, and they would adjust to that situation. The next section showcases the implementation details of such an approach.

4.2 Application of Evolutionary Algorithms

To be able to achieve the self-adaptation needed from our dashboard, a mechanism for handling the fittest metrics is needed [36, 37] [1]. The goal can be derived from the fitness function, which we define as follows:

definition

Definition 1. *An agent has a higher contribution/fitness to the overall system if and only if a slight change in a specific metric would yield a significant change in the amount of energy saved, the difference between the expected energy efficiency level and the current level diverge significantly, or a metric answers custom energy-related questions that users may have.*

With this in mind, by applying the fitness function to each of the metrics inside the system we would get a numeric score for each of them. This tells the system how much does each individual metric influence the overall energy efficiency level of system as a whole. Furthermore, the user would be shown only the metrics particularly relevant to the current state of the system. Thus, by choosing this exact fitness function, these would follow:

- metrics that deviate from the mean are more likely to be shown to the user;
- metrics whose improvement may result in other metrics' fitness are more likely to be shown to the user;
- metrics of all sizes whose improvement would not yield a substantial increase/decrease in energy efficiency levels, have less change of being shown to the user.

The final algorithm would then run as follows:

1. **assign** a fitness score to each agent in the system (according to the fitness function defined above)

[1] Notice the use of the word *fittest*, indicating the usage of natural selection as a mean of choosing the metrics that tell the most about the specified goal.

2. **select** members to act upon using some variation operators (crossover and mutations) [38] [2]
3. **replace** certain members of the population with these children from variation operators
4. **keep** some members from the previous population in the new population

The next step would be then to perform the *natural selection* between the metrics. More specifically, we chose the tournament-like contest in which the winner would continue to breed, and the loser would be most likely eliminated [39, 40].

4.3 Dynamic Equilibrium.

As energy efficiency monitoring solutions became larger, stakeholders operating such systems would have significant difficulties determining which metric to display and optimize for, and which to ignore. Such a system is said to be deterministic, as all the change has to be manually done and the system has no means of improvement.

Advances in the field of big data and machine learning indicate that the above-mentioned system architecture could be transformed to be stochastic, with no absolute governance. In addition, the data being collected is growing exponentially [41], and thus more metrics are going to become available for processing. A non-deterministic system operating in such a way would neither be in the state of maximum entropy, nor would it be completely stable. Such a state is referred to as *Dynamic Equilibrium*, where actors are loosely bound to each other while still having plenty of room to improve.

5 Future Research Agenda

To date, research on software-oriented energy efficiency has been focusing almost solely on code-related metrics, such as those presented in section 2. However, no evidence proves the importance of process-related metrics (productivity, turnaround time, throughput, iteration burn-down, error rate, etc) to the energy efficiency. Thus, our research agenda includes primary research to investigate the correlation of above-described metrics to the energy consumption and saving levels.

In addition, we aim to extend the list of energy consumption code metrics considered by the dashboard. For example by knowing the Number of Classes (NoC) in the source code we can easily calculate other metrics like: Number of Immediate sub-classes of a Class, Response for a Class, Weighted methods per Class as well as the Number of Children. This may lead to performance benefits and ultimately savings in computing power. Performance is one of the key aspects that is also a part of our research agenda.

[2] Crossover is combination of parents' genetic information, and mutation is a change in agent's genetic information

Finally, we would like to supplement metrics with a reliable prediction system [42–45].

6 Conclusions

Energy efficiency, the minimization of the amount of energy required to provide products and services, has become a growing issue. Mobile phones, smart watches and other portable devices operate in a very power-constrained environment. Research in the area of energy efficiency would facilitate the efforts to cut energy costs, and ultimately reduce greenhouse gas emissions [46].

Advent of software solutions, on the other hand, have profoundly impacted almost all spheres of human existence. To date, more attention has been given to the study of energy efficiency in hardware, rather than in software [47]. This paper considered code, as well as process and project-related metrics as indicators of potential energy savings. Additionally, we presented a dashboard that tracks such metrics and only displays the ones that are most relevant. It manages to do so by combining the notions from Complexity Theory (Non-linear dynamics, Dynamic Equilibrium) with Evolutionary Algorithms (natural selection) to synthesize an adaptable stochastic tool.

7 Acknowledgments

The work presented in this paper was supported by the grant of Russian Science Foundation # 19-19-00623.

References

1. Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31. ACM, 2014.
2. Gustavo Pinto and Fernando Castor. Energy efficiency: a new concern for application software developers. *Communications of the ACM*, 60(12):68–75, 2017.
3. Kenan Liu, Gustavo Pinto, and Yu David Liu. Data-oriented characterization of application-level energy optimization. In *International Conference on Fundamental Approaches to Software Engineering*, pages 316–331. Springer, 2015.
4. Stephen Few. Information dashboard design. 2006.
5. Stephen Few and Perceptual Edge. Dashboard confusion revisited. *Perceptual Edge*, pages 1–6, 2007.
6. Shadan Malik. *Enterprise dashboards: design and best practices for IT*. John Wiley & Sons, 2005.
7. Emanuele Danovaro, Tadas Remencius, Alberto Sillitti, and Giancarlo Succi. PKM: Knowledge Management Tool for Environments Centered on the Concept of the Experience Factory. In *Companion of the 30th International Conference on Software Engineering*, ICSE Companion '08, pages 937–938. ACM, 2008.
8. Andrea Janes, Alberto Sillitti, and Giancarlo Succi. Effective dashboard design. *Cutter IT Journal*, 26(1), 2013.

9. Vladimir Ivanov, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasily Zorin. Precooked Developer Dashboards: What to Show and How to Use – Poster. In *Proceedings of the 40th International Conference on Software Engineering Companion*, ICSE'18, Gothenburg, Sweden, May-June 2018. ACM.
10. Vladimir Ivanov, Vladislav Pischulin, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasilii Zorin. Design and validation of precooked developer dashboards. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, pages 821–826, 2018.
11. Irina D Coman, Pierre N Robillard, Alberto Sillitti, and Giancarlo Succi. Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software*, 91:124–134, 2014.
12. Andrea Janes and Giancarlo Succi. *Lean Software Development in Action*. Springer, Heidelberg, Germany, 2014.
13. Witold Pedrycz, Barbara Russo, and Giancarlo Succi. A model of job satisfaction for collaborative development processes. *Journal of Systems and Software*, 84(5):739–752, 2011.
14. Witold Pedrycz, Barbara Russo, and Giancarlo Succi. Knowledge transfer in system modeling and its realization through an optimal allocation of information granularity. *Appl. Soft Comput.*, 12(8):1985–1995, August 2012.
15. Frank Maurer, Giancarlo Succi, Harald Holz, Boris Kötting, Sigrid Goldmann, and Barbara Dellen. Software Process Support over the Internet. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 642–645. ACM, May 1999.
16. Alberto Sillitti, Tullio Vernazza, and Giancarlo Succi. Service Oriented Programming: A New Paradigm of Software Reuse. In *Proceedings of the 7th International Conference on Software Reuse*, pages 269–280. Springer Berlin Heidelberg, April 2002.
17. Luis Corral, Alberto Sillitti, and Giancarlo Succi. Software Assurance Practices for Mobile Applications. *Computing*, 97(10):1001–1022, October 2015.
18. Young Choon Lee and Albert Y Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.
19. György L Kovács, Sylvester Drozdik, Paolo Zuliani, and Giancarlo Succi. Open Source Software for the Public Administration. In *Proceedings of the 6th International Workshop on Computer Science and Information Technologies*, October 2004.
20. Brian Fitzgerald, Jay P Kesan, Barbara Russo, Maha Shaikh, and Giancarlo Succi. *Adopting open source software: A practical guide*. The MIT Press, Cambridge, MA, 2011.
21. Enrico Di Bella, Alberto Sillitti, and Giancarlo Succi. A multivariate classification of open source developers. *Information Sciences*, 221:72–83, 2013.
22. Ding Li and William G. J. Halfond. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software - GREENS 2014*. ACM Press, 2014.
23. Alexander Chatzigeorgiou and George Stephanides. *Software Quality Journal*, 10(4):355–371, 2002.
24. Cristea Vlad Vasile, Colin Pattinson, and Ah-Lian Kor. Mobile phones and energy consumption. In *Green IT Engineering: Social, Business and Industrial Applications*, pages 243–271. Springer International Publishing, September 2018.

25. Ching Kin Keong, Koh Tieng Wei, Abdul Azim Abd. Ghani, and Khaironi Yatim Sharif. Toward using software metrics as indicator to measure power consumption of mobile application: A case study. In *2015 9th Malaysian Software Engineering Conference (MySEC)*. IEEE, December 2015.
26. Luis Cruz and Rui Abreu. Performance-based guidelines for energy efficient mobile applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, May 2017.
27. Tullio Vernazza, Giampiero Granatella, Giancarlo Succi, Luigi Benedicenti, and Martin Mintchev. Defining Metrics for Software Components. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, volume XI, pages 16–23, July 2000.
28. Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Measures for mobile users: an architecture. *Journal of Systems Architecture*, 50(7):393–405, 2004.
29. Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. A Relational Approach to Software Metrics. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1536–1540. ACM, 2004.
30. Stephen Few. Dashboard design: Taking a metaphor too far. *Information Management*, 15(3):18, 2005.
31. Stephen Few and Perceptual Edge. Data visualization: past, present, and future. *IBM Cognos Innovation Center*, 2007.
32. John O Louch, Eric Steven Peyton, Christoper Hynes, Scott Forstall, and Gregory N Christie. Synchronization of widgets and dashboards, October 22 2013. US Patent 8,566,732.
33. Murray G Patterson. What is energy efficiency?: Concepts, indicators and methodological issues. *Energy policy*, 24(5):377–390, 1996.
34. Anthony J Corrado. *Dynamics of complex systems*. CRC Press, 2019.
35. John Michael Tutill Thompson, Michael Thompson, and Hugh B Stewart. *Nonlinear dynamics and chaos*. John Wiley & Sons, 2002.
36. SN Sivanandam and SN Deepa. Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer, 2008.
37. Stephanie Forrest. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.
38. Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.
39. Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In *ICGA*, volume 95, pages 9–15. Citeseer, 1995.
40. GW Greewood, Gary B Fogel, and Manuel Ciobanu. Emphasizing extinction in evolutionary programming. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 666–671. IEEE, 1999.
41. Ioannis Anagnostopoulos, Sherali Zeadally, and Ernesto Exposito. Handling big data: research challenges and future directions. *The Journal of Supercomputing*, 72(4):1494–1516, 2016.
42. Petr Musílek, Witold Pedrycz, Nan Sun, and Giancarlo Succi. On the Sensitivity of COCOMO II Software Cost Estimation Model. In *Proceedings of the 8th International Symposium on Software Metrics*, METRICS '02, pages 13–20. IEEE Computer Society, June 2002.
43. Marco Ronchetti, Giancarlo Succi, Witold Pedrycz, and Barbara Russo. Early estimation of software size in object-oriented environments a case study in a cmm level 3 software firm. *Information Sciences*, 176(5):475–489, 2006.

44. Bruno Rossi, Barbara Russo, and Giancarlo Succi. Modelling Failures Occurrences of Open Source Software with Reliability Growth. In *Open Source Software: New Horizons - Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems, OSS 2010*, pages 268–280, Notre Dame, IN, USA, May 2010. Springer, Heidelberg.
45. Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. Understanding the Impact of Pair Programming on Developers Attention: A Case Study on a Large Industrial Experimentation. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1094–1101, Piscataway, NJ, USA, June 2012. IEEE Press.
46. Len Brookes. The greenhouse effect: the fallacies in the energy efficiency solution. *Energy policy*, 18(2):199–201, 1990.
47. Eugenio Capra, Chiara Francalanci, and Sandra A Slaughter. Measuring application software energy efficiency. *IT Professional*, 14(2):54–61, 2012.