Integrated Energy-Directed Test Suite Optimization

Ding Li, Yuchen Jin
Computer Science Department
University of Southern California
Los Angeles, CA, USA
{dingli, yuchenji}@usc.edu

Cagri Sahin, James Clause
Computer and Information Sciences Department
University of Delaware
Newark, DE, USA
{cagri, clause}@udel.edu

William GJ Halfond
Computer Science Department
University of Southern California
Los Angeles, CA, USA
halfond@usc.edu

ABSTRACT

In situ testing techniques have become an important means of ensuring the reliability of embedded systems after they are deployed in the field. However, these techniques do not help testers optimize the energy consumption of their in situ test suites, which can needlessly waste the limited battery power of these systems. In this work, we extend prior techniques for test suite minimization in such a way as to allow testers to generate energy-efficient, minimized test suites with only minimal modifications to their existing work flow. We perform an extensive empirical evaluation of our approach using the test suites provided for real world applications. The results of the evaluation show that our technique is effective at generating, in less than one second, test suites that consume up to 95% less energy while maintaining coverage of the testing requirements.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Experimentation, Measurement

Keywords

Energy usage, Test suites, Minimization

1. INTRODUCTION

Ensuring the continued reliability of software running on embedded systems is challenging. These systems operate with severe resource constraints and in diverse and rapidly changing environments. This can lead to unexpected failures that only manifest in the field due to changing environmental conditions, hardware component failure, or unexpected events [6,17,38,41]. Therefore, developing validation or "self-check" tests that run routinely on a deployed system has become a necessary step to ensure the system's continuing correct behavior.

Recognizing this need, researchers have developed techniques for testing, analyzing, and debugging software systems in situ (e.g., [20, 25, 30]). These techniques help software engineers ensure that their software and the underlying systems are running correctly in the field. However, these techniques neglect an import concern: energy consumption. This is especially important because the in-situ test suites are run on the device on a periodic basis. As a result, this self-check process comes with an energy cost, and testers must strike a balance between thoroughly testing a system's behavior and preserving its limited battery power. In the worst case, the test suites can needlessly drain the limited battery power of the underlying hardware by running redundant and expensive tests, and thus prevent the system from carrying out its required functionality [10].

In practice, testers lack techniques to help them appropriately balance energy consumption and test requirements coverage. As a result, they are often forced to use ad hoc techniques to attempt to save energy. Unfortunately, such ad hoc attempts are often ineffective and, in the worst case, can lead to increased, rather than decreased, energy consumption. For example, a tester may try to minimize the energy consumption of an in situ test suite by using existing minimization tools (e.g., [8, 14]) to select the fewest number of tests such that the selected tests maintain the coverage level of the original test suite. As we show in Section 3 and Section 5, this approach can be wildly off the mark and result in the creation of test suites that consume far more energy than necessary. This leads to sub-optimal usage of a system's limited battery power, which, in turn, can lead to reduced system capabilities and higher maintenance costs due to battery recharging or replacement.

In this paper, we present and evaluate an improved version of our approach, the Energy-directed Test Suite Optimizer (EDTSO), for minimizing the energy consumption of in-situ test suites. Our approach allows software testers to combine energy minimization with traditional minimization goals [33], such as maintaining fault finding capability and minimizing test suite size. To do this, EDTSO encodes the test suite minimization problem as an integer linear programming (ILP) problem. Solving the ILP problem then results in a minimized test suite that is guaranteed to satisfy the tester's minimization goals and use as little energy as possible. In prior work, we presented a preliminary version of EDTSO and an initial case study of its effectiveness [19].

ISSTA '14, July 21–25, 2014, San Jose, CA, USA

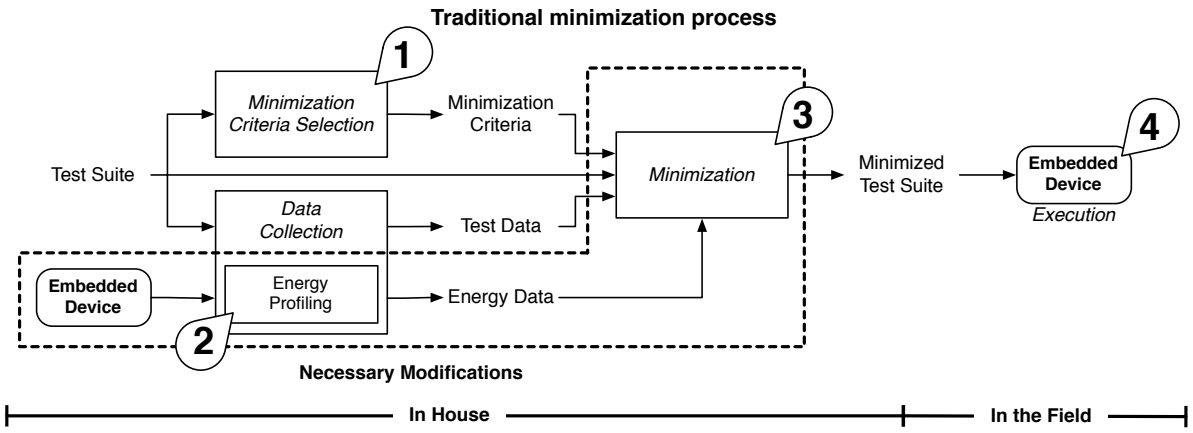

Figure 1: Overview of the in situ test suite minimization process.

We have extend the EDTSO approach so that it can (1) be integrated into existing test workflows, and (2) utilize other types of resource constraints, such as execution time. We also performed an extensive empirical evaluation of EDTSO based on the developer provided test suites of real-world applications. Our evaluation also investigates how EDTSO's performance is impacted when it uses more easily collected proxy measurements, such as execution time, as a proxy for directly measuring the energy usage of test cases.

The results of our evaluation show that EDTSO is able to create, in a reasonable amount of time, minimized test suites that always consume less energy than the corresponding test suites created by a traditional, size-focused technique. Moreover, the amount of savings is significant; in some cases, the test suites produced by EDTSO consumed over 95 % less energy than the original test suite. Our investigation into the feasibility of using execution time as a proxy for energy measurements shows that, although execution time could be useful, directly considering energy consumption results in consistently higher savings. Overall, we believe that these are strong results and indicate that EDTSO can become a vital tool for embedded systems developers, as it will allow them to perform more thorough post-deployment validation without compromising the functionality of their applications.

The remainder of this paper is organized as follows: Section 2 provides background information about testing embedded systems and test suite minimization. In Section 3, we present a motivating example that we use to illustrate our approach and also the results of a motivating study. Section 4 describes the details of our approach. Section 5 presents our prototype implementation and the empirical evaluation. Section 7 discusses related work. Finally, Section 8 presents our conclusions and planned future work.

2. BACKGROUND

This section provides necessary background information on the key differences between testing embedded systems and testing traditional software as well as the current workflow testers use to minimize their test suites.

2.1 Testing Embedded Systems

At a high-level, the process of testing embedded systems is similar to that for traditional software (e.g., desktop applications). *First*, the tester identifies the requirements that the system under test must satisfy. *Second*, the tester creates a set of tests (i.e., a test suite) that is capable of determining whether the system satisfies its requirements. *Finally*, the tests are run and the results evaluated. This process is performed by testers of embedded and traditional software before the system is deployed to end users.

A key difference in the embedded systems domain is that testers typically create additional test suites, one for each embedded device on which the application will be deployed. These additional test suites are often referred to as "self-check", "self-diagnostic", or in situ test suites as they are deployed with the system and run periodically to ensure that both the software and system are working correctly. The testing requirements for such test suites are typically functionally oriented. For example, they may test a particular sensor or the network connectivity of the system. These tests ensure that the different features of the software system are still functioning as the environment containing the system changes. Since the tester knows a priori the details of the embedded system on which the in-situ test suites will run, they will often attempt to optimize each in-situ test suite, with respect to its corresponding embedded device, in order to minimize the test suite's resource usage. It is this optimization step that EDTSO automates.

2.2 Test Suite Minimization Workflow

To optimize their in situ test suites, testers often perform some type of *test suite minimization*. Essentially, they are selecting, in a principled manner, a subset of the original, complete test suite that will be executed—a minimized test suite. Figure 1 provides a high-level overview of the process for creating minimized test suites.

The first step in the process, *Minimization Criteria Selection*, is to determine the criteria that will be used to decide whether or not to include a specific test in the minimized test suite. The choice of whether a test should be included can be based on many factors. For example, testers may choose to include tests based on their age (e.g., how long ago they were written), how effective they are (e.g., how many failures they have detected in the past), how long they take to execute, etc., or based solely on their intuition. However, regardless of the factors used to make the decisions, the overall goal remains the same: to create a test suite that is as small

Table 1: Motivating example.

Coverage	Test Cases				
	t_1	t_2	t_3	t_4	t_5
1. public void updateData(Location prior) {	✓	✓	✓	✓	✓
2. Location current = LocationManager.getLocation();	✓	✓	✓	✓	✓
3. SensorData data = getCurrentData();	✓	✓	✓	✓	✓
4. if (prior.distanceTo(current) > MAX_DISTANCE) {	✓	✓	✓	✓	✓
5. data = Sensors.getData(current);		✓		✓	
6. }	✓	✓	✓	✓	✓
7. if (OutOfDate(data)) {	✓	✓	✓	✓	✓
8. data = Sensors.getData(current)			✓		✓
9. }	✓	✓	✓	✓	✓
10. if (TimePassed() > MAX_TIME) {	✓	✓	✓	✓	✓
11. resendMessage(data);	✓			✓	✓
12. }	✓	✓	✓	✓	✓
13. }	✓	✓	✓	✓	✓
Energy Consumption (mJ)	0.5	0.7	0.2	1.3	1.5

as possible but also as effective as the original test suite.

The second step in the process, *Data Collection*, is to gather the necessary *test-related data*. That is the data about each test in the test suite that is needed to decide whether each test meets the minimization criteria chosen in the first step. For example, if the chosen minimization criteria include test case age, then it is necessary to identify when each test case in the test suite was written. Similarly, if the minimization criteria include effectiveness, then it is necessary to know how many failures each test case has revealed in the past.

The third step in the process, *Minimization*, is to create the minimized test suite by judging each test case with respect to the chosen minimization criteria. In practice, this is often the most difficult step as choosing the "best" minimized test suite is an NP-Hard problem (it can be reduced to the set cover problem). To facilitate the creation of "good" minimized test suites, researchers have formalized the test suite minimization problem and provided a variety of tools for creating minimized test suites (see [43] for an extensive list of such techniques).

Finally, the last step in the testing process, *Execution*, deploys the minimized test suite to the device where it is repeatedly executed. Note that this is the only part of the minimization process that actually takes place in the field; *Minimization Criteria Selection*, *Data Collection*, and *Minimization* are all conducted in house.

3. MOTIVATION

In this section we introduce a running example that serves to illustrate our approach and highlight some of the energy inefficiencies that can occur when minimization focuses on test suite size instead of energy consumption. We also present the results of an exploratory study that further motivates our approach by investigating the energy consumption of real-world test cases.

3.1 Illustrative Example

To illustrate how testers may inadvertently create minimized test suites that consume more energy than necessary, consider the example application excerpt shown in Table 1. This code obtains the system's current location (Line 2) and checks to see if the system has moved more than a certain distance from its prior location (Line 4). If so, the system gets the sensor data about its current location (Line 5). The application also checks to see if the sensor data is out of date and if so reperforms the sensor measurements (Line 8). Finally, if more than a certain amount of time has passed since the device last sent out a beacon message, the message is resent with the new data (Line 11).

Table 1 also shows coverage information for the application's test suite. The columns under the heading *Test Cases* show, for each test case in the test suite, whether it covers each statement in the application. A check mark (✓) indicates that a statement is covered by a test case while a blank space indicates that a statement is not covered. For example, Line 11 is covered by test cases t_1, t_4, and t_5, but not covered by t_2 or t_3. Taken together, t_1 through t_5 achieve 100 % statement coverage for the application.

Assume that a tester wants to minimize this test suite before deploying it to run in situ. More specifically, assume that the tester wants to create an energy-minimized test suite that maintains the same level of statement coverage as the original test suite (i.e., the minimized test suite should also cover every line in the program). For our example, there are several subsets of the original test suite that achieve 100 % statement coverage. Consider two such subsets. The first is t_1, t_2, and t_3, which we will refer to as test suite T_1. The second is t_4 and t_5, which we will refer to as test suite T_2. Both T_1 and T_2 achieve the same coverage as the original test suite, but with fewer test cases.

T_2 is more likely to be produced by existing minimization tools as it contains fewer test cases than T_1. However, choosing T_2 does not satisfy the tester's desire that the minimized test suite consume as little energy as possible. To illustrate why this is the case, consider the energy costs associated with each test case, which are shown in the final row in Table 1. These numbers show the amount of energy, in milli-Joules (mJ), consumed by each test case when it executes on the target embedded system.

By summing the energy cost of each test case in T_1 and T_2, we can calculate the test suites' total energy costs of 1.4 mJ and 2.8 mJ, respectively. As can be seen, even though T_2 contains fewer test cases than T_1, it consumes twice as much energy. This difference can be attributed to the fact that in T_2 both test cases power up the 3G radio to send the mes-

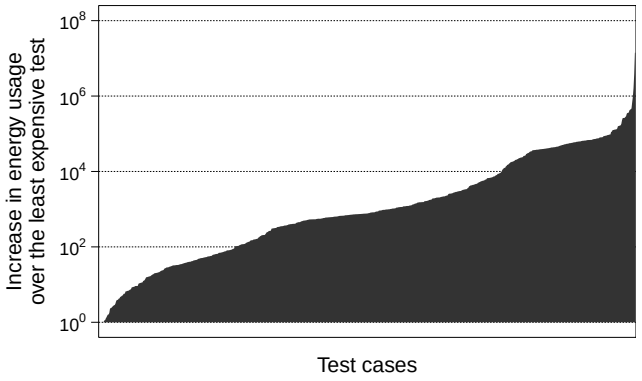

Figure 2: Ratio of the energy usage of a test to the energy usage of the least expensive test in the test's test suite.

sage (Line 11)—an operation that requires a large amount of energy—while T_1 only includes one test case (t_1) that powers up the 3G radio. Although a difference of 1.4 mJ may not be significant in absolute terms, as we explained above, in situ test suites are generally executed repeatedly. This amplifies the impact of such a difference multiplicatively.

3.2 Exploratory Study

To further motivate our proposed work, we performed an exploratory study that looked at the energy usage of real-world test cases. In particular, we investigated: (1) whether there is a large variance in the energy usage of test cases within a test suite, (2) whether there is a large variance in the energy usage of test suites that achieve approximately the same level of coverage, and (3) how much energy could potentially be lost by minimizing test suites with respect to their size rather than their energy usage. To carry out this study, we analyzed 767 test cases that comprise the developer-created test suites of seven real world applications. Details of these applications and their test suites are given in Section 5.2 and Table 2. To gather the necessary data, we measured the energy usage and statement coverage of each test case using the methodology outlined in Section 5.3.

We first compared the energy consumption of the test cases within a test suite. To do this we normalized the energy usage measurements by dividing the energy usage of each test case by the energy usage of the test case in its test suite that consumes the least amount of energy. Using the running example to illustrate, the energy usages of $t_1, t_2, \ldots, t_5$ would be divided by 0.2 mJ, the energy usage of t_3. Figure 2 shows the results of this computation for the 767 test cases we considered. The y-axis shows the normalized energy usages on a logarithmic scale and the x-axis shows each test, sorted from left to right in increasing order of normalized energy usage.

As Figure 2 shows, the energy costs of individual test cases can vary greatly, even among test cases in the same test suite. On average, the test cases in each suite consume approximately 75,000 times more energy than the least expensive test case. Moreover, in the most extreme cases, the energy consumption of the most expensive test cases is approximately 14 million times more expensive than the least expensive test case. Based on this figure, we can see that test cases are likely to have a high amount of variance in their energy usages.

We next compared the energy usage of test suites that achieve approximately the same amount of statement coverage. Since our subject applications only have one test suite each, we generated multiple test suites by randomly selecting subsets of each application's original test suite. In total, we generated 70,000 unique test suites, 10,000 for each subject. We then calculated what percentage of the original test suite's statement coverage was achieved by each random test suite and, based on this calculation, binned the random test suites into 10 coverage groups: (0 %–10 %], [10 %–20 %], ..., [90 %–100 %]. Finally, we calculated the energy usage of each randomly generated test suite.

Figure 3 shows seven, Tukey-style box plots, one for each subject, that summarize the results of this analysis. In each box plot, the x-axis shows the coverage groups and the y-axis shows the percentage reduction in energy usage compared to the subject's original test suite. (This was done to normalize energy consumption.) In each individual box, the horizontal line and the upper and lower edges show the median and the upper and lower quartiles, respectively, for the percentage of energy used at each coverage level. As the figure shows, for almost all of the coverage groups there is a wide range in the amount of energy consumed by the test suites. This result shows that the potential impact of minimizing test suites with respect to energy usage can be quite large. In fact, for several coverage levels, the difference between the most efficient and least efficient test suites is nearly 90 %.

A certain amount of variance among both test cases and test suites is to be expected, since tests access expensive system resources in different patterns. However, the amount of variance exhibited in Figures 2 and 3 demonstrates that the most common approach for test suite minimization (i.e., selecting the smallest test suite possible) is often sub-optimal when energy consumption is a concern; selecting a larger number of energy-inexpensive test cases may result in a minimized test suite that consumes less energy.

Finally, we investigated how much energy may be wasted by choosing minimized test suites based on size rather than energy usage. Within each of the coverage groups, we compared the test suite that consumes the least amount of energy against the smallest test suite. Over 75 % of the time the test suite that consumes the least amount of energy is not the smallest test suite. Moreover, on average, the least expensive test suite consumes ≈18 % less energy than the smallest test suite, despite being ≈11 % larger.

Overall, the results of this study motivate our approach by showing (1) there is a large amount of variance in the energy usage of test cases within the same test suite, (2) there is a large amount of variance in the energy usage of test suites that achieve approximately the same level of coverage, and (3) choosing the smallest test suite is likely to result in unnecessary energy usage.

4. APPROACH

The goal of our approach is to enable testers to produce energy efficient minimized in situ test suites. To do this, we define a technique, based on test suite minimization approaches, that selects test cases based not only on their coverage of a software system's test requirements but also on their energy usage. The design of our technique allows testers to integrate it with the existing test suite minimization process and optimize their test suites with respect to

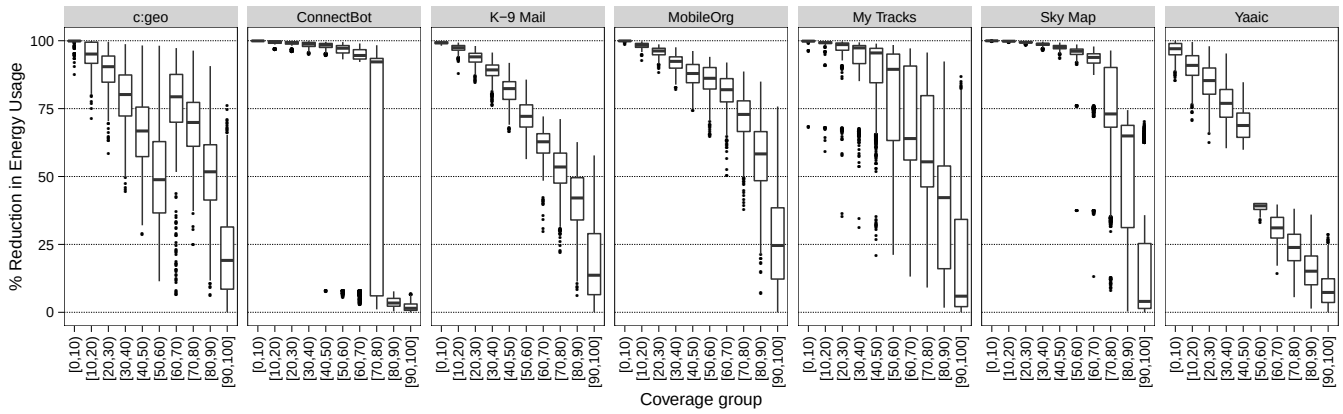

Figure 3: Variability in the energy consumption of test suites within a coverage group.

```
 1. bin: b1, b2, b3, b4, b5;

 2. min: 0.5 b1 + 0.7 b2 + 0.2 b3 + 1.3 b4 + 1.5 b5;

 3.  s1: b1 + b2 + b3 + b4 + b5 >= 1;
 4.  s2: b1 + b2 + b3 + b4 + b5 >= 1;
 5.  s3: b1 + b2 + b3 + b4 + b5 >= 1;
 6.  s4: b1 + b2 + b3 + b4 + b5 >= 1;
 7.  s5:      b2 +      b4 + b5 >= 1;
 8.  s6: b1 + b2 + b3 + b4 + b5 >= 1;
 9.  s7: b1 + b2 + b3 + b4 + b5 >= 1;
10.  s8:           b3 +      b5 >= 1;
11.  s9: b1 + b2 + b3 + b4 + b5 >= 1;
12. s10: b1 + b2 + b3 + b4 + b5 >= 1;
13. s11: b1 +           b4      >= 1;
14. s12: b1 + b2 + b3 + b4 + b5 >= 1;
15. s13: b1 + b2 + b3 + b4 + b5 >= 1;
```

Figure 4: ILP encoding of our motivating example.

energy-efficiency as easily as they can optimize with respect to traditional criteria, such as fault detection capability. The remainder of this section describes (1) our approach for creating energy-efficient, minimized test suites, and (2) how our approach integrates into the existing test suite minimization process described in Section 2.

4.1 Energy Minimization

Our technique for creating energy-efficient, minimized test suites is based on encoding test case selection as a constrained minimization problem. Here the constraints are the minimization criteria chosen by a tester in the *Minimization Criteria Selection* step, and the minimization objective is the amount of energy consumed by the minimized test suite. The formulation of the minimization problem relies on the fact that test suite minimization can be expressed as an ILP problem (e.g., [8,14]). ILP is commonly used to compute an optimal solution to a mathematical problem, given a set of constraints on the solution. The primary benefit of using an ILP-based approach is that the solution to the minimization problem (i.e., a minimized test suite) is guaranteed to satisfy the minimization criteria, as long as the problem is solvable. Although ILP problems are NP-Complete, we have found that, in practice, all of the minimization problems we addressed were solvable in less than one second (see Section 5.5).

To encode a minimization problem as an ILP problem, the technique relies on three insights [14]. First, a minimized test suite T' for a test suite T can be represented as an array of binary values $A = \left[b_1, b_2, \ldots, b_{|T|}\right]$. A value of true (resp., false) for b_i indicates that the ith test case in T should (resp., should not) be included in T'. Second, minimizing an objective function $\sum_i^{|T|} e_i b_i$, where e_i is the energy usage of test case t_i, also minimizes the energy usage of T'. And finally, minimization criteria can be encoded as linear relationships among the elements of A.

To illustrate the process of encoding a minimization problem as an ILP problem, consider again Table 1. Recall that in this example, the tester wants to create a minimized test suite that maintains the same level of statement coverage as the original test suite while minimizing energy consumption. The corresponding ILP problem is shown in Figure 4, using a standardized linear programming syntax [2]. In the figure, Line 1 shows each of the test cases represented as an array of binary values $b_1, b_2, \ldots, b_5$, one for each of the five test cases. Line 2 defines the objective function, where each binary variable is weighted by the energy cost of the corresponding test case, and Lines 3–15 define the constraints on the solution. Because the minimization criteria requires 100 % statement coverage, there is one constraint for each of the 13 statements (lines) in the application. For example, Line 13 expresses the fact that, for Statement 11 to be covered, the minimized test suite must include either test case t_1 or t_4. Similarly, Line 15 indicates that selecting any test case will result in a minimized test suite that covers Statement 13.

Once the encoding is complete, the approach uses an ILP solver to generate a solution to the ILP problem. For this step, any standard ILP solver can be used. The solution to the problem shown in Figure 4 assigns true to variables b_1, b_2, and b_3. Since the variables correspond to test cases, the selection of the test cases follows in a straightforward manner. The minimized test suite that results from this process is t_1, t_2, and t_3, which is the energy efficient test suite T_1 identified in Section 3. The test suite can then be deployed to the energy constrained device and run in the field as part of the normal execution step described in Section 2.

4.2 Integration

The design of our approach allows it to easily integrate into existing test suite minimization processes. In fact, our

approach only requires two minor changes to the process outlined in Section 2. In Figure 1, the necessary modifications are shown using a dashed box. The first, and simplest, change is that during the *Minimization* step, testers would use our approach's test suite minimization technique instead of their current minimization tool. Our approach can support any minimization criteria, as long as they can be expressed as a system of constraints. This includes all widely used testing criteria such as white-box (e.g., statement, branch, path, etc.) and black-box (e.g., requirements, use cases, etc.). So by using our approach, testers do not lose any capabilities. Additional information about how such criteria can be encoded can be found in prior work (e.g., [8, 14, 19]).

The second change is that, during the *Data Collection* step, the testers must collect additional information about the energy usage of each test in the original test suite. More specifically, they must create a mapping $E : T \rightarrow \mathbb{R}$ that maps each test case $t \in T$ to its energy cost when executed on the target embedded device. Since energy cost can vary from device to device depending on each device's specific features, it is necessary to calculate E and run the EDTSO process for each target device. We assume that these devices are known to developers a priori and are accessible to them for this purpose. The energy information is used to weight the variables in the objective function as described in Section 4.1. As we show in Section 5.6, EDTSO performs most effectively using the actual energy costs of each test cases. However, if energy usage measurements are unavailable, more easily gathered proxy metrics, such as execution time, can be substituted.

To create the mapping from test cases to energy usage, the approach (1) instruments the test suite to log the times when each test case starts and stops executing (e.g., insert a call to `System.out.println(System.currentTimeMillis());`), and (2) measures the amount of energy consumed by each test case on the target embedded system using an energy measurement platform (EMP). We assume that testers have sufficient access to either the binary or source of the software system for instrumentation purposes since this part of the testing process is performed in-house.

Instrumentation of a test suite can be fully automated in cases where the test suite is developed using a framework, such as JUnit or TestNG. Since test suites are defined using a standardized method, automatically identifying the points where a test case starts and stops executing is a straightforward process. For example, in JUnit 3 tests follow strict naming conventions and in JUnit 4 and TestNG tests are identified by an annotation (e.g., `@Test`). In cases where the test suite is written in an ad hoc manner, a tester must either write an instrumenter to automatically insert instrumentation in the correct places or they must manually insert probes into the test suite. Regardless of how instrumentation is added, once the test suite is instrumented it is executed using an EMP.

At a high-level, an EMP is simply a mechanism for calculating the amount of energy consumed by executing a piece of code on a specific platform. In practice, there are many ways that such energy usage information can be calculated, including hardware-based approaches that use physical instrumentation and monitoring (e.g., [18, 34, 36, 39]), simulation-based approaches that replicate the actions of a processor and estimate energy consumption of each executed cycle by using a cycle-accurate simulator (e.g., [9, 12, 24]), and estimation-based approaches that model energy-influencing features to estimate energy usage (e.g., [4, 13, 23, 26]). In general, the output of an EMP is a timestamped log that records how much power is being consumed. By combining the log with the timestamps provided by the instrumentation, it is straightforward to calculate the individual energy usage of each test case. In our implementation of EDTSO, we run each test on a hardware-based EMP that we constructed by attaching a power monitor to the embedded system. More details about our chosen EMP are provided in Section 5.3.

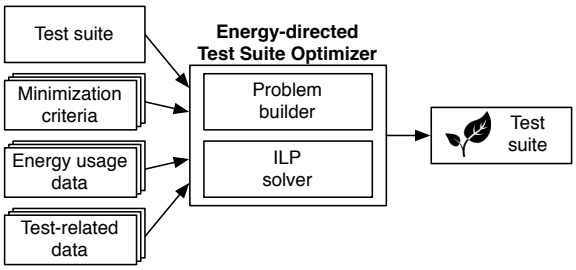

Figure 5: Overview of the Energy-directed Test Suite Optimizer.

To illustrate the modified *Data Collection* step, consider again the example shown in Table 1. The input to the *Data Collection* step is the test suite T, which contains $\{t_1, t_2, \ldots, t_5\}$. The approach instruments T, executes it, and measures the energy usage of each t_i. For example, suppose that a test case t_i starts at time $s(t_i)$ and ends at time $e(t_i)$. Its energy consumption, $E(t_i)$, is the total energy consumed between $s(t_i)$ and $e(t_i)$. The output of the step is the function E where each test case is mapped to the energy value shown at the bottom of Table 1. For example, $E(t_2) \rightarrow 0.7$ and $E(t_5) \rightarrow 1.5$.

5. EVALUATION

We investigated the usefulness and effectiveness of EDTSO in an empirical evaluation. For the evaluation, we created a prototype implementation of EDTSO and used it to address the following research questions:

RQ1: Usefulness. Do test suites generated by EDTSO use less energy than test suites generated by a traditional, size-focused minimization approach?

RQ2: Minimization Time. How much time does EDTSO need to generate energy minimized test suites?

RQ3: Proxy Measures. How is EDTSO's performance impacted by using proxy measures instead of energy usage measurements?

5.1 Prototype Tool

Figure 5 presents a high-level overview of the implementation of EDTSO. As the figure shows, the tool takes as input a *test suite*, *minimization criteria*, *test-related data*, and *energy data*. Its output is the energy-minimized test suite that will be executed in situ.

The *test suite*, *test-related data*, and *minimization criteria* are the information needed by the existing test suite minimization process described in Section 2. The *energy data*

Table 2: Subject applications.

Subject	Description	LoC	# Tests
c:geo	Geocashing client	51,451	176
ConnectBot	SSH client	50,851	22
K-9 Mail	Email client	71,816	38
MobileOrg	Org-mode file manipulation	14,819	87
MyTracks	Track user paths	35,039	310
Sky Map	Astronomical map	21,121	134
Yaaic	IRC client	19,293	28

is the energy cost of executing each test case on a platform of interest (see Section 4.2). The *problem builder* is responsible for encoding the inputs into an ILP problem (see Section 4.1). And finally, the *ILP solver* is responsible for solving the ILP problem generated by the problem builder. For our experiments, we used lp_solve [1].

5.2 Embedded System Selection

An embedded system is comprised of both a hardware platform and application software that runs on that platform. For our experiments we chose to use an Android-based system as our hardware platform, since Android is rapidly becoming a common operating system for many embedded systems (e.g., [11, 16, 35, 42]). More specifically, we chose to use a Low Power Energy Aware Processing (LEAP) node [36]. Our LEAP node is an embedded x86 platform based on an ATOM N550 processor that runs Android 3.2. The sensor configuration of the LEAP node is similar to that used in other embedded systems [37, 40]. The use of the LEAP node offers many benefits. First, as compared to other embedded systems, such as the REF TEK earthquake monitor [3], it is relatively inexpensive. Second, many Android systems make their applications and test suites publicly available, allowing us to draw from a broad pool of applications that represent different testing and implementation practices. Finally, we have extensive infrastructure to run Android applications, measure the energy usage of their tests, and collect coverage data.

For the software, we selected the applications shown in Table 2. The first two columns, *Subject* and *Description*, list the name of each application and a brief description of its functionality. The third column, *LoC*, shows the application's number of lines of code. The final column, *# Tests*, shows the number of test cases included in each application's test suite. All of our subject applications are available on Google's Play App Store.

5.3 Experiment Protocol

For each of the subjects, we collected data about the test cases in each test suite. For each test case, we measured its structural coverage, energy usage, and execution time.

Structural coverage. To obtain structural coverage data, we instrumented each application using a technique that is based on Ball and Larus' approach for efficient path profiling [5]. This technique allows us to minimally and efficiently place probes in an application and determine which bytecode instructions were covered during an execution.

Energy consumption. To measure the amount of energy consumed by a test case, we ran the instrumented application on the LEAP node. The LEAP node is connected to an analog-to-digital data acquisition (DAQ) card that samples current draw for each component (e.g., WiFi, GPS, memory, and CPU) at 10 kHz. Because the DAQ is external to the LEAP, it does not introduce any measurement overhead to the system. However, the instrumentation that is added to the application has an execution cost, which we subtract from the final energy numbers. To determine the overhead, we profiled the energy cost of the inserted instrumentation. Since we know exactly how many times and when the instrumentation is executed, we can accurately remove its overhead from our energy measurements.

Execution time. As we explained in Section 4.2, the instrumentation added to the test suite records the start and stop times of each test case. The difference between these values is the execution time of the test case.

5.4 RQ1: Effectiveness

To address the first research question, we compared the energy consumption of test suites minimized using EDTSO against the energy consumption of test suites minimized using a traditional, size-focused technique. To do this, we first generated 70,000 ILP minimization problems, 10,000 for each subject. To generate each minimization problem, we randomly chose a subset of the testing requirements (i.e., statements that must be covered) and used EDTSO and the size-focused technique to create a minimized test suite that satisfies the chosen requirements (i.e., covers the necessary statements). Then we calculated and compared the energy consumption of the resulting test suites.

Table 3 and Figure 6 show a summary of this comparison. Table 3 shows how often the energy-minimized test suite consumed strictly *less* energy than the size-minimized test suite. Note that in the remaining cases, the energy usages of the energy-minimized and size-minimized suites are the same; EDTSO never created a minimized test suite that consumed more energy than the test suite created by the size-focused minimization technique. For each subject, we present the overall results and also break down the results by coverage group. Here each test suite's coverage group is determined by the percentage of statements required to be covered by the corresponding minimization problem.

Figure 6 shows, for the cases when the energy-minimized test suites consume less energy than the size-minimized test suite, a Tukey-style boxplot of the percentage improvement in energy usage of the energy-minimized test suites over the size-minimized test suites. As in Table 3, the results are again broken down by subject and coverage group. Note that these numbers represent *in the field* energy savings. We do not take into account the energy cost of producing the minimized test suite as this part of the process occurs in house, prior to deploying the test suite to the embedded system. In addition, note that these savings are for a *single execution* of the test suite.

The results in Table 3 show that, across all applications, the energy-minimized test suites consume less energy than the size-minimized test suites over 92 % of the time. The amount of improvement was also significant. The overall mean improvement in energy usage ranges from 3.8 % to 17.9 % and the overall maximum improvement was always nearly 100 %. These results are encouraging as they demonstrate that minimizing test suites with respect to their energy usage not only consistently provides energy savings, but also that the amount of savings is significant.

By looking more closely at the data, some interesting trends emerge. *First*, EDTSO is less effective when mini-

Table 3: Percentage of the time that test suites generated using EDTSO consume less energy than test suites generated using a traditional, size-focused technique.

	% Better						
Coverage	c:geo	ConnectBot	K-9 Mail	MobileOrg	My Tracks	Sky Map	Yaaic
[0,10)	74	55	40	74	76	69	50
[10,20)	95	48	56	67	92	85	77
[20,30)	99	58	70	80	99	98	88
[30,40)	100	71	85	95	99	100	96
[40,50)	100	68	90	95	100	99	99
[50,60)	100	69	94	95	100	100	49
[60,70)	100	62	87	97	100	100	77
[70,80)	100	58	93	100	100	100	90
[80,90)	100	61	94	100	100	100	97
[90,100]	100	62	99	100	100	100	99
Overall	99	63	90	98	99	99	88

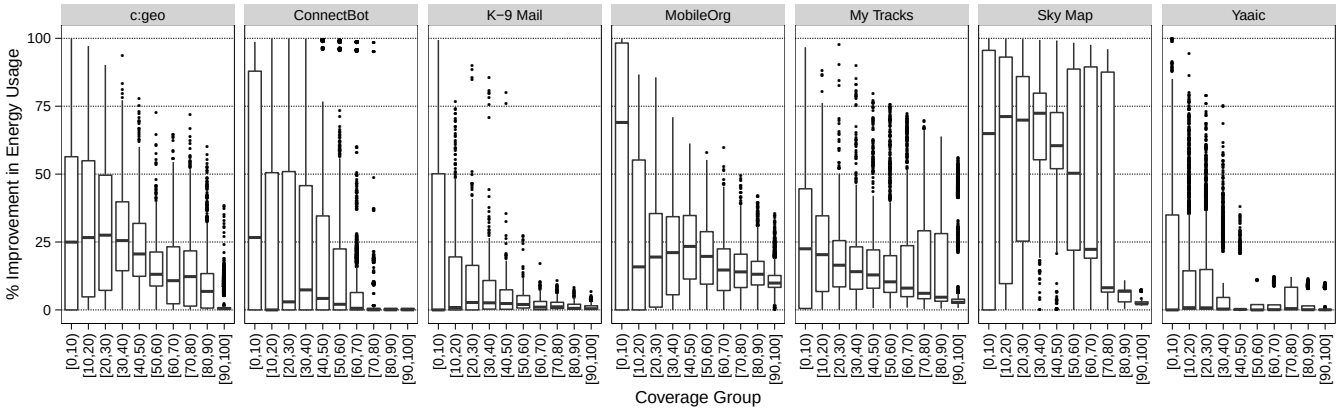

Figure 6: Percentage improvement in energy usage of test suites generated using EDTSO over test suites generated using a traditional, size-focused technique.

mizing smaller test suites. The results show that EDTSO is less effective, both in terms of the number of times it can save energy and the amount of energy that is saved, for ConnectBot, K-9 Mail and Yaaic, our subjects with the smallest test suites. Intuitively, this makes sense as larger test suites are more likely to have test cases with overlapping coverage, which allows for more ways to satisfy the minimization criteria. *Second*, from Table 3, it appears that it is more difficult to produce energy efficient test suites at lower coverage levels. As the coverage levels increase, the percentage of the time EDTSO created a more efficient test suite also increases. We hypothesize that this is not a generalizable trend, but instead is tied to the fact that the subjects' test suites generally contain test cases that, individually, cover large portions of the applications. As a result, it is likely that the same test cases are included in both the energy-minimized and size-minimized test suites. *Third*, from Figure 6 we can see that the percentage improvement in energy usage decreases as the coverage level increases. Again, we believe that this is due to the composition of the test suites. As the required coverage approaches 100 %, there are simply fewer ways to find subsets of the original test suite that also get the same coverage. Therefore, there is less room for improvement at higher coverage levels.

5.5 RQ2: Minimization Time

The purpose of our second research question is to determine whether EDTSO can generate minimized test suites in a reasonable amount of time. We judged reasonableness in two ways. *First*, we looked at the execution times of EDTSO in absolute terms. Looking at execution times in absolute terms is useful because it can help us to decide whether the technique is likely to be used in practice, and if so, in what contexts. For all 70,000 minimization problems that we considered, EDTSO was able to generate an energy-efficient, minimized test suite in less than 1 s. *Second*, we looked at the execution times of EDTSO in relative terms by determining whether there is a significant difference in the execution times of the two techniques (using the Mann-Whitney-Wilcoxon Test) and, if so, the size of the effect (using Vargha and Delaney's $\hat{A}_{12}$ statistic). Yaaic is the only subject where there is a significant difference in execution time ($p < 0.05$). However, the effect size is small ($\hat{A}_{12} = 0.56$) which indicates that the improvement is slight. Overall, the results of these comparisons are promising: there is no execution time penalty to using EDTSO, and the speed of EDTSO will add little overhead to the testing or deployment processes.

5.6 RQ3: Proxy Measures

Our results from RQ1 show that EDTSO can consistently produce test suites that are more energy efficient than those generated by a size-focused technique. However, in some

situations it may not be possible for testers to obtain energy data. For example, estimation-based EMPs may not be available for their target platform and hardware-based EMPs may be too expensive or difficult to use. For this reason, we also investigated whether it was possible to achieve energy savings with our approach using proxy measures instead of energy. For this experiment we used the execution time of each test case as a proxy for energy usage, as execution time is commonly perceived to be positively correlated with energy usage and is easy for testers to collect.

To create test suites that are minimized with respect to execution time, we created a modified version of EDTSO that used execution time, rather than energy usage, as the objective function's constants. This modified version was used to create time-minimized test suites by solving the 70,000 minimization problems we created for RQ1. We then compared the energy usage of the time-minimized test suites with the energy usage of the energy-minimized test suites created for investigating RQ1.

Table 4 and Figure 7 show a comparison between the effectiveness of EDTSO when using energy measurements and the effectiveness of EDTSO when using execution time measurements. Note that the formatting used for Table 4 and Figure 7 is the same as for Table 3 and Figure 6. In all cases, the amount of energy consumed by the energy-minimized test suites was less than or equal to that of the time-minimized test suites. Therefore, the table and figure again show data only for the cases where the energy-minimized test suite consumed strictly *less* energy than the time-minimized test suite.

The data in Table 4 and Figure 7 give mixed results for the effectiveness of using execution time as a proxy for energy usage. On the positive side, the average loss in energy efficiency is low, ranging from 0.1 % to 5.3 %. This indicates that, in the average case, a tester could expect to see similar energy savings when using execution times as a proxy. However, the maximum savings show that a tester could miss significant savings by not using energy measurements. We also investigated whether there was a significant difference in the execution times of the tools and found that there was no difference between using energy measurements and execution time measurements. From these results, we conclude that using energy costs provides a superior solution in terms of maximizing energy savings. However, if energy costs are unavailable, then it is still possible to achieve energy savings with our approach using execution time as a proxy measure.

6. THREATS TO VALIDITY

External Validity: The applications that need in situ testing are usually sensor and network intensive applications, which makes them different from many normal Android applications. To represent these applications, we selected seven real world applications. All of these seven applications have extensive network usage and two of them, c:geo and MyTracks, also have heavy GPS usage. Overall, the functionalities of the seven applications in our evaluation were similar to the applications that would require in situ testing.

Construct Validity: The energy savings of our approach depend on the quality of the test suites. However, it may not be appropriate to assume that the real in situ test suites have the same quality as the test suites of our seven applications. To address this problem, we generated many new test suites for our applications by randomly selecting subsets of the original test suites with different coverage levels. Each generated test suite is essentially a new test suite for the application and could be evaluated independently. Since the new test suites are randomly generated, we avoid introducing any assumptions about the quality of the original test suites.

7. RELATED WORK

The work presented in this paper extends the previous version of our technique [19]. There are several conceptual and practical differences between the version of EDTSO presented in this paper and the version presented in prior work. First, we have carried out experiments to provide background and motivational data that illustrates current energy consumption trends with real world applications. Second, we have redesigned our technique so that it can integrate into existing testing workflows. Third, the evaluation of the prior work was a small case study, whereas this paper contains a more extensive empirical evaluation with seven test suites created by the developers of real world applications. Fourth, we evaluated EDTSO using time-based resource data and compared the effectiveness of such an approach with our own energy based approach.

In addition to our own prior work, researchers have investigated techniques for detecting energy bugs (e.g., [21,28,29, 31]) and for using expensive resources effectively (e.g., [15, 27]). The most related of these techniques investigates using the dynamic voltage and frequency scaling (DVFS) capabilities of certain processors to reduce the energy consumption of regression tests [15]. At a high-level, they are using DVFS to throttle the CPU to an appropriate level, as determined by analyzing previous executions of the regression tests. Note that, unlike our approach, the regression test suite is executed in its entirety. As such, our techniques are complimentary; a combined approach could both create energy-efficient minimized test suites and use DVFS to potentially achieve even greater energy savings.

Also related to our technique is existing work on test suite minimization, also known as test case selection. Such work can be divided into several high-level categories [32]: *Retest all techniques*, which simply rerun all existing test cases. *Random or ad hoc techniques*, which randomly select an arbitrary percentage of the existing test cases. *Minimization techniques*, which select the minimum number of test cases necessary to achieve a given criterion. For example, a minimization technique could select the smallest number of test cases such that all added or modified statements in the new version are executed. *Safe techniques* select every existing test case that achieves a certain criterion. For example, a safe technique could select every test case that executes an added or modified statement in the new program. *Pareto efficiency techniques*, which optimize test suites based on multi-objective criteria. Yoo and Harman provide a recent, detailed survey of existing work in each of these categories [43].

Because the types of techniques mentioned above are designed for regression testing, they should not be seen as alternatives to our approach, but rather complimentary. Our approach takes the concept of test suite optimization and introduces the new idea of using energy consumption as the limiting criteria.

Table 4: Percentage of the time that test suites generated by EDTSO using energy measurements consume less energy than test suites generated by EDTSO using execution time measurements.

	% Better						
Coverage	c:geo	ConnectBot	K-9 Mail	MobileOrg	My Tracks	Sky Map	Yaaic
[0,10)	8	0	25	52	48	15	28
[10,20)	14	0	42	52	56	26	66
[20,30)	22	7	57	62	63	46	82
[30,40)	27	29	74	79	81	42	94
[40,50)	26	43	83	84	88	19	99
[50,60)	15	41	89	84	97	2	31
[60,70)	18	18	78	90	97	2	65
[70,80)	24	2	88	96	99	7	84
[80,90)	20	0	91	98	100	4	96
[90,100]	5	0	98	100	100	0	99
Overall	15	17	76	94	95	7	84

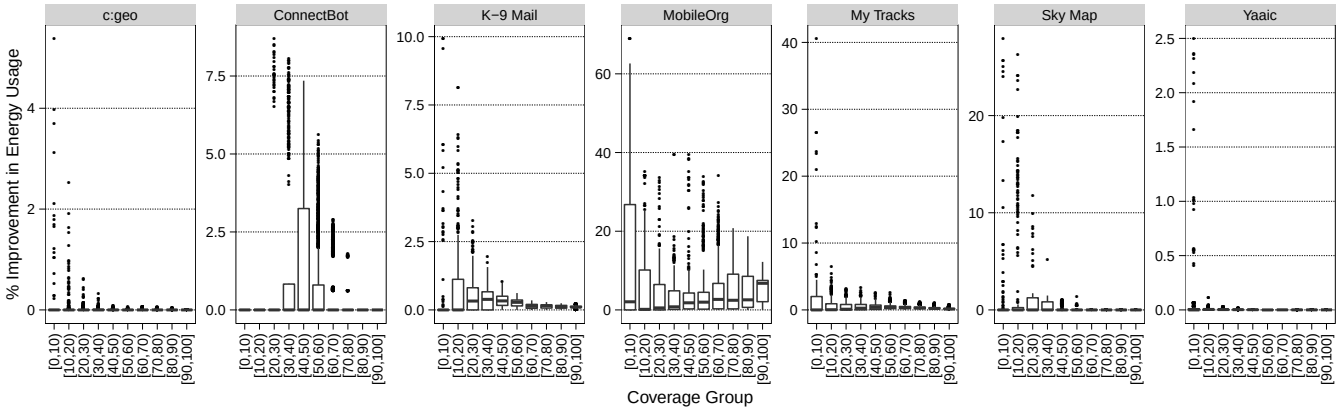

Figure 7: Percentage improvement in energy usage of test suites generated by EDTSO using energy measurements over test suites generated by EDTSO using execution time measurements.

In addition to test suite minimization techniques, our approach is also related to work on post deployment validation, or more generally, techniques that leverage data collected from deployed software (e.g., [7, 20, 22, 25, 30, 44]). Like the test suite minimization techniques described above, techniques that gather information from the field are not alternatives to our technique. Instead, they represent opportunities where our general approach of optimizing with respect to energy usage can be applied.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we extended our prior work on creating energy-efficient test suites that can be used to perform post-deployment testing on embedded systems. Our approach integrates with existing test suite minimization processes and allows testers to minimize in situ test suites based on energy usage in addition to traditional criteria. It is based on casting the test suite minimization problem as an ILP problem. We implemented our approach in a prototype tool and used it to conduct an empirical evaluation on the test suites of seven real world applications. The results of the evaluation indicate that our approach can consistently generate, in a reasonable amount of time, minimized test suites that consume up to 95 % less energy than test suites generated by traditional, size-focused techniques. The evaluation also demonstrates that our technique is most effective when using energy measurements rather than using execution time as a proxy. Overall, we believe that these results are very positive and indicate that our approach can help developers of embedded software reduce the energy consumption of their test suites.

We plan on investigating several areas of future work. First, we will extend the evaluation of EDTSO. More specifically, we will consider more applications and test suites as well as additional embedded systems. Expanding the evaluation will allow for additional confidence in the generality of our results. Second, we will investigate the effects of using other structural coverage metrics, such as branch coverage, on the performance of the technique. Third, we will investigate the performance of alternative constraint solvers. Although lp_solve was able to handle all of the minimization problems that we considered, alternative solvers may perform better on larger problems. Information about the relative performance of ILP solvers can help testers choose which to use in practice. Fourth, we will create and study techniques for determining the leading causes of energy consumption in a test suite. These techniques will focus on identifying what types of tests consume the most amount of energy. Finally, we will consider the use of search-based strategies to solve the minimization problems. Compared to ILP-based formulations, search-based systems can often provide additional information (e.g., Pareto-optimal fronts) that can help testers choose between comparable solutions.

9. REFERENCES

[1] lpsolve.sourceforge.net.

[2] lpsolve.sourceforge.net/5.5/lp-format.htm.

[3] www.reftek.com/.

[4] N. Amsel and B. Tomlinson. Green tracker: A tool for estimating the energy consumption of software. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems: Extended Abstracts*, pages 3337–3342, 2010.

[5] T. Ball and J. R. Larus. Efficient path profiling. In *Proceedings of the 29th annual ACM/IEEE International Symposium on Microarchitecture*, pages 46–57, 1996.

[6] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, pages 43–56, 2008.

[7] A. Bertolino, G. De Angelis, and A. Sabetta. VCR: Virtual capture and replay for performance testing. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 399–402, 2008.

[8] J. Black, E. Melachrinoudis, and D. Kaeli. Bi-criteria models for all-uses test suite reduction. In *Proceedings of the 26th International Conference on Software Engineering*, pages 106–115, 2004.

[9] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual International Symposium on Computer Architecture*, pages 83–94, 2000.

[10] Z. Chen and K. G. Shin. Post-deployment performance debugging in wireless sensor networks. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 313–322, 2009.

[11] T. Chikaraishi, T. Minato, and H. Ishiguro. Development of an android system integrated with sensor networks. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 326–333. IEEE, 2008.

[12] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The SoftWatt approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pages 141–151, 2002.

[13] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating mobile application energy consumption using program analysis. In *Proceedings of the 35th International Conference on Software Engineering*, pages 92–101, 2013.

[14] H.-Y. Hsu and A. Orso. MINTS: A general framework and tool for supporting test-suite minimization. In *Proceedings of the 31st International Conference on Software Engineering*, pages 419–429, 2009.

[15] E. Y. Kan. Energy efficiency in testing and regression testing – a comparison of DVFS techniques. In *Proceedings of the 13th International Conference on Quality Software*, pages 280–283, 2013.

[16] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12(2):74–82, 2011.

[17] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, pages 174–174, 2006.

[18] D. Li, S. Hao, W. G. Halfond, and R. Govindan. Calculating source line level energy information for android applications. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, July 2013.

[19] D. Li, C. Sahin, J. Clause, and W. G. Halfond. Energy-directed test suite optimization. In *Proceedings of the 2nd International Workshop on Green and Sustainable Software*, pages 62–69, 2013.

[20] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 15–26, 2005.

[21] Y. Liu, C. Xu, and S. Cheung. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In *Proceedings of the 2013 IEEE International Conference on Pervasive Computing and Communications*, pages 2–10, 2013.

[22] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 343–351, 2011.

[23] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual International Conference on Mobile Computing and Networking*, pages 317–328, 2012.

[24] T. Mudge, T. Austin, and D. Grunwald. The reference manual for the Sim-Panalyzer, version 2.0. http://web.eecs.umich.edu/~panalyzer/.

[25] C. Murphy, G. Kaiser, I. Vo, and M. Chu. Quality assurance of software applications using the in vivo testing approach. In *Proceedings of the 2nd International Conference on Software Testing Verification and Validation*, pages 111–120, 2009.

[26] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. A preliminary study of the impact of software engineering on GreenIT. In *Proceedings of the First International Workshop on Green and Sustainable Software*, pages 21–27, 2012.

[27] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *Proceedings of the 8th international Conference on Mobile Systems, Applications, and Services*, pages 299–314, 2010.

[28] A. Pathak, Y. C. Hu, and M. Zhang. Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, pages 5:1–5:6, 2011.

[29] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: Characterizing

and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 267–280, 2012.

[30] C. Pavlopoulou and M. Young. Residual test coverage monitoring. In *Proceedings of the 21st International Conference on Software Engineering*, pages 277–284, 1999.

[31] B. Priyantha, D. Lymberopoulos, and J. Liu. LittleRock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, pages 12–15, 2011.

[32] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions Software Engineering*, 22(8):529–551, 1996.

[33] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, 1997.

[34] C. Sahin, F. Cayci, I. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh. Initial explorations on design pattern energy usage. In *Proceedings of the First International Workshop on Green and Sustainable Software*, pages 55–61, 2012.

[35] C. Seeger, A. Buchmann, and K. Van Laerhoven. myhealthassistant: A phone-based body sensor network that captures the wearer's exercises throughout the day. In *Proceedings of the 6th International Conference on Body Area Networks*, pages 1–7. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[36] D. Singh, P. A. H. Peterson, P. L. Reiher, and W. J. Kaiser. *The Atom LEAP platform for energy-efficient embedded computing: Architecture, operation, and system implementation*, 2010. `http://lasr.cs.ucla.edu/leap/FrontPage?action=AttachFile&do=get&target=leapwhitepaper.pdf`.

[37] R. Smith, J. Das, H. Heidarsson, A. Pereira, F. Arrichiello, I. Cetnic, L. Darjany, M.-E. Garneau, M. Howard, C. Oberg, M. Ragan, E. Seubert, E. Smith, B. Stauffer, A. Schnetzer, G. Toro-farmer, D. Caron, B. Jones, and G. Sukhatme. USC CINAPS builds bridges. *Robotics & Automation Magazine, IEEE*, 17(1):20–30, 2010.

[38] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 51–63, 2005.

[39] watts up. `https://www.wattsupmeters.com/secure/index.php`.

[40] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 108–120. IEEE, 2005.

[41] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 381–396, 2006.

[42] J. Whipple, W. Arensman, and M. S. Boler. A public safety application of gps-enabled smartphones and the android operating system. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 2059–2061. IEEE, 2009.

[43] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[44] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 91–100, 2009.