Do Search-Based Approaches Improve the Design of Self-Adaptive Systems ? A Controlled Experiment

Sandro Santos Andrade
GSORT Distributed Systems Group
Federal Institute of Education, Science,
and Technology of Bahia. Salvador-Ba. Brazil
Email: sandroandrade@ifba.edu.br

Raimundo José de Araújo Macêdo
Distributed Systems Laboratory (LaSiD)
Department of Computer Science
Federal University of Bahia. Salvador-Ba. Brazil
Email: macedo@ufba.br

***Abstract*—Endowing software systems with self-adaptation capabilities has shown to be quite effective in coping with uncertain and dynamic operational environments as well as managing the complexity generated by non-functional requirements. Nowadays, a large number of approaches tackle the issue of enabling self-adaptive behavior from different perspectives and under diverse assumptions, making it harder for architects to make judicious decisions about design alternatives and quality attributes trade-offs. It has currently been claimed that search-based software design approaches may improve the quality of resulting artifacts and the productivity of design processes, as a consequence of promoting a more comprehensive and systematic representation of design knowledge and preventing design bias and false intuition. To the best of our knowledge, no controlled experiments have been performed to provide sound evidence of such claim in the self-adaptive systems domain. In this paper, we report the results of a quasi-experiment performed with 24 students of a graduate program in Distributed and Ubiquitous Computing. The experiment evaluated the design of self-adaptive systems using a search-based approach, in contrast to the use of a style-based non-automated approach. The results show that search-based approaches can improve the effectiveness of resulting architectures and reduce design complexity. We found no evidence regarding the method's potential for leveraging the acquisition of distilled design knowledge by novice software architects.**

I. Introduction

Increasingly demands for guaranteeing scalability, dependability, energy-efficiency, and performance in dynamic environments have challenged the way we develop software systems. Elastic data storage services, energy-aware mobile systems, self-tuning databases, and reconfigurable network services are some of the application domains in which self-adaptive mechanisms play a paramount role [1]. Such scenarios are usually characterized by incomplete knowledge about user requirements, workloads, and available resources. As a consequence, committing to a particular solution in design time may yield suboptimal architectures, which easily degrade the service when conditions deviate from those previously defined.

A self-adaptive (SA) system continuously monitor its own behavior and its operating environment, adapting itself whenever current conditions prevent it from delivering the expected quality of service [2]. SA systems usually comprise two parts: a managed element and a managing element [3]. The managed element provides functional services to the user, operating in a potentially dynamic and uncertain environment. The managing system is responsible for adapting the managed element, mostly by using a particular implementation of an adaptation loop. The MAPE-K approach [4] is a widely accepted reference architecture for adaptation loops. It defines the basic components for the loop's tasks of **M**onitoring, **A**nalyzing, **P**lanning, and **E**xecuting; performed with the support of a **K**nowledge Base.

The many approaches for self-adaptation available nowadays adopt different mechanisms for the aforementioned tasks. Reflexive middleware platforms, graph grammars, intelligent agents, policy-based approaches, self-organizing structures, and control theory are some of the currently adopted underpinnings for enabling self-adaptation [5]. Such large solution space, along with the intricate problem space that characterizes the SA systems domain, make it harder for architects to judiciously evaluate all available design alternatives and make well-informed decisions on quality attributes trade-offs. As a consequence, false intuition, design bias, time-to-market constraints, and/or incomplete knowledge about the solution space may lead to suboptimal architectures, which do not fully realize the self-adaptation requirements at hand.

Over the past twelve years, Search-Based Software Engineering (SBSE) [6] has provided promising approaches for tackling the aforementioned issues in areas such as requirements engineering, design, testing, and refactoring, just to mention a few. SBSE claims that the majority of issues in such areas are indeed optimization questions and that the software's virtual nature is inherently well suited for search-based optimization [7]. In particular, substantial work towards search-based software design [8] advocate the benefits of SBSE in finding out subtle effective designs and providing well-informed means to reveal quality attributes trade-offs.

To the best of our knowledge, the first effort in applying search-based approaches to the design of SA systems is that proposed by us in [9]–[11]. In such work, we provide a meta-modeling infrastructure for defining domain-specific design spaces which systematically capture the domain's prominent design dimensions, their associated variation points (alternative solutions), and the architectural changes required to realize each solution. The goal is to support the automated redesign of an initial model, endowing it with additional capabilities from the application domain at hand. Each domain-specific design space entails a set of quality metrics that evaluate each candidate architecture regarding different attributes.

We have been using such approach to enable the automated

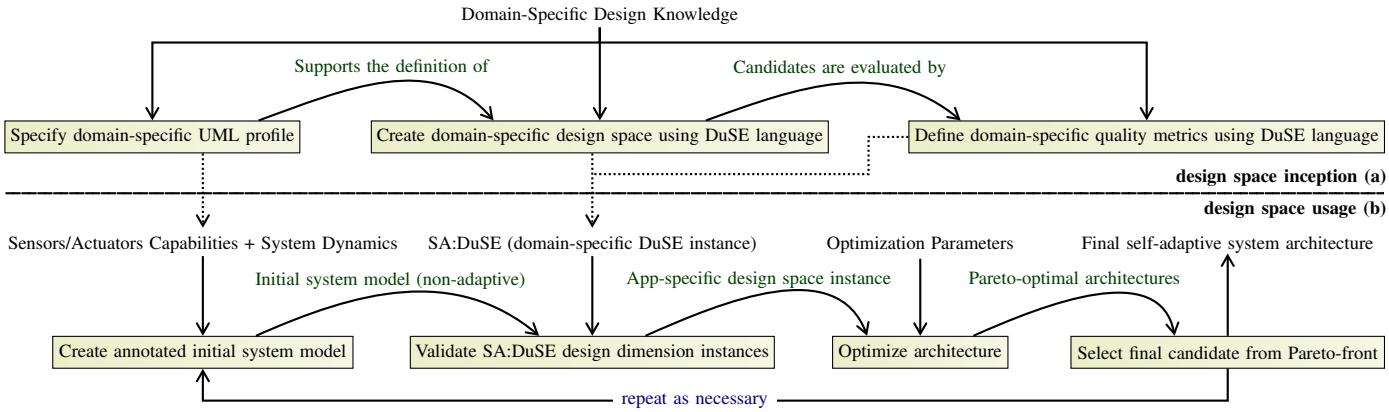

Fig. 1. Overview of our architecture design approach. In the *design space inception* stage (a), domain experts use the DuSE's constructs for specifying a domain-specific design space (degrees of freedom, their corresponding alternative solutions, and the architecture extensions required to reify each solution). Henceforth – in the *design space usage* stage (b) – architects submit initial models to the optimization engine, which searches for those (near-)optimal extensions revealing design trade-offs. A particular design space instance – SA:DuSE – enables the use of such infrastructure in the self-adaptive systems domain.

design of managing elements for initial (non-adaptive) systems such as web servers and MapReduce distributed architectures [12]. Since even small input models usually span huge design spaces, we also provide a domain-independent multi-objective optimization engine. Such engine currently relies on the NSGA-II algorithm [13] to find out a set of Pareto-optimal [14] candidate architectures. All these solutions represent optimal architectures, differing only in which quality metric they favor.

In this paper, we report the results of a quasi-experiment [15] performed with 24 students of a graduate program in Distributed and Ubiquitous Computing. We evaluated the design of SA systems using our search-based approach, in contrast to using a style-based non-automated approach. The experiment goal was to **analyze** the design of SA systems, **for the purpose of** evaluating the search-based design approach we propose and a design process based on architecture styles catalogs, **with respect to** the effectiveness and complexity of resulting architectures, as well as the method's potential for leveraging the acquisition of distilled design knowledge by novice SA systems architects, **from the viewpoint of** researchers, and **in the context of** graduate students endowing systems with self-adaptation capabilities.

The quasi-experiment is characterized as a blocked subject-object study with a paired comparison design. Two UML models representing a web server and a MapReduce distributed architecture are used as experiment objects and two treatments (search-based approach and style-based approach) are considered for the design method factor (independent variable). We are interested in evaluating the impact of the adopted design method on three dependent variables: the effectiveness and complexity of resulting architectures, as well as the method's potential for leveraging the acquisition of distilled design knowledge by novice architects. We use the Generational Distance metric [14], [16] to assess effectiveness in terms of how far the architectures designed by the subjects are from a previously determined Pareto-optimal set of architectures. Design complexity is evaluated by using the Component Point approach [17] while a questionnaire with multiple choice questions evaluates the method's potential for leveraging the acquisition of design knowledge. All the material used in the experiment is available at http://wiki.ifba.edu.br/tr-ce012014.

The remainder of this paper is organized as follows. Section II presents an overview of the experiment. Section III explains the experiment objects, the hypotheses being investigated, the adopted measurement approach, and the experiment design. In section IV we analyze and discuss the results, while threats to validity are identified in section V. Finally, section VI presents related work while conclusions and venues for future work are presented in section VII.

II. Experiment Definition

Designing effective architectures for SA systems is a challenging task. It requires architects become familiar with the intricacies of both the problem space (so that accurate and realistic self-adaptation requirements can be elicited) and solution space (in order to adopt the most effective adaptation strategy/mechanism for the problem at hand). That involves deciding on self-adaptation goals; system and environment monitoring mechanisms; measurement noises and uncertainties; unanticipated/unforeseen adaptations; diverse control robustness degrees; change enacting mechanisms; and adaptation temporal predictability, just to mention a few [1], [18], [19].

A number of efforts from the software engineering for SA systems community [5] have addressed the issue of providing principled engineering approaches, leveraging the systematic capture of design knowledge and enabling the early reasoning of self-adaptation quality attributes. Our approach – named DuSE – joins such endeavor by providing: *i)* an infrastructure for systematically representing distilled architecture design knowledge for a given application domain (design space); *ii)* a domain-independent architecture optimization engine as the underlying mechanism for explicitly eliciting design trade-offs (conflicting quality attributes); and *iii)* a concrete design space for the SA systems domain.

As depicted in Fig. 1, a concrete design space and its quality attributes are specified by experts once per application domain (*design space inception* stage) by using the DuSE modeling language. A supporting UML profile is also defined for that domain, enabling the annotations that drive the automated design process. A *design space* (e.g., for networked and concurrent systems) is defined as a set of n *design dimensions*

representing specific design concerns in such a domain (e.g., concurrency strategy and event dispatching model).

Each design dimension entails a set of *variation points*, representing alternative solutions for such a concern (e.g., leader-followers or half-sync/half-async; for the concurrency strategy dimension). A variation point describes the elements (*architectural extensions*) that must be added to the initial model in order to realize such particular solution. Therefore, a candidate architecture (a location in such n-dimensional space) is formed by the initial model extended with the merge of all architectural extensions provided by all involved variation points. *Validation rules*, defined per variation point, support the automatic detection of invalid architectures. Once a concrete design space is defined, architects can submit initial models to manual design space exploration or rely on the multi-objective optimization engine we provide (*design space usage* stage).

Each design dimension holds an OCL expression that relies on the associated UML profile's annotations to detect, in the initial model, the architectural *loci* that demand decisions about such concern. For instance, an initial model may require the choice of particular control strategies for two different service components. Therefore, two instances of the control strategy design dimension are created to capture the decisions for those architectural *loci*. As a consequence, huge design spaces may easily be spawned even for small input models, motivating the adoption of meta-heuristics and multi-objective optimization approaches. The domain-independent optimization engine we provide handles all required steps to forge candidate architectures for a given set of design space locations, evaluate their quality regarding the attributes defined for the design space, and find out a set of Pareto-optimal architectures. Further information about the DuSE meta-model and its architecture optimization engine may be found in [9], [10].

The aforementioned infrastructure provides the underpinnings of our SA systems design approach. We have specified a particular DuSE instance (SA:DuSE) that captures the most prominent degrees of freedom and quality attributes when designing adaptation loops based on feedback control [20]. Currently, SA:DuSE yields architectural extensions regarding six different control laws (P, PI, PID, static state feedback, precompensated static state feedback, and dynamic state feedback), seven empirical tunning approaches (four Chien-Hrones-Reswick variations, Ziegler-Nichols, Cohen-Coon, and LQR), three mechanisms for control adaptation (gain-scheduling, model-reference, and model-identification), and three different control loops deployment arrangements (global, local control + shared reference, and local control + shared error). In addition, four quality metrics related to control overhead, average settling time, average maximum overshoot, and control robustness are also available. Due to space limitations, we omit here a thorough description of SA:DuSE dimensions, its corresponding variation points, and quality metrics. Further information may be found in [10].

Our approach has been fully implemented in a supporting tool named DuSE-MT[1], developed using the C++ programming language and the Qt cross-platform toolkit[2]. DuSE-MT is a meta-model agnostic tool we develop in order to support general software modeling activities and, in particular, the automated design process we propose. The NSGA-II evolutionary algorithm [13] is currently used as optimization backend, but other approaches can be easily adopted in the future thanks to the DuSE-MT's plugin-based architecture and the optimization engine's internals we have designed.

[1]http://duse.sf.net

[2]http://qt-project.org

TABLE I. OVERVIEW OF THE 32H COURSE IN WHICH THE EXPERIMENT WAS UNDERTAKEN.

Part	Day	Activity
Lectures	1	Self-Adaptive Systems Foundations (motivation, MAPE-K reference architecture, current approaches, challenges)
	2	Feedback Control Introduction (control goals, control properties, fixed gain SISO approaches)
	3	Feedback Control (MIMO and adaptive approaches)
	4	Self-Adaptive Systems - Case Studies
Exam and Training	5	First hour: discussion Next 3h: Pen and paper exam
	6	First hour: exam discussion Next 3h: Training (DuSE-MT and architectural styles catalog)
Experiment	7	First 110min: Tests #1 and #2 Next 110min: Tests #3 and #4 Next 20min: Tests #5 and #6 (questionnaire)
	8	First 110min: Tests #7 and #8 Next 110min: Tests #9 and #10 Next 20min: Tests #11 and #12 (questionnaire)

We have been comparing the quality of MapReduce elastic architectures generated by our approach with real implementations of such solutions in a cluster running Apache Hadoop v2.3[3]. The goal is to evaluate to which extent the predicted quality properties are indeed observed in real prototypes. As a complementary evaluation effort, in this work we look for any empirical evidence supporting the claim that search-based approaches improve the effectiveness and reduce the complexity of SA systems architectures. Furthermore, we want to know whether search-based approaches leverage the acquisition of distilled design knowledge by novice architects.

III. EXPERIMENT PLANNING

The experiment took place as part of a 32 hours course on Software Engineering for Distributed Systems, arranged in eight classes (four hours each) along four weeks. As presented in Table I, the course was split in three parts: lectures, exam and training, and experiment.

In the first four classes, students were exposed to the foundations of SA systems and feedback control, as well as to the SISO (Single-Input Single-Output) and MIMO (Multiple-Input Multiple-Ouput) feedback control strategies [20] more widely adopted in SA systems. It is worth mentioning that all students had previously undertaken a 32 hours course on Software Architecture and Software Modeling. Roughly half of them work as software developers/designers, while the remaining have a stronger background in network administration. We try to insulate the effect of this factor by using blocking techniques as described in subsection III-D. Furthermore, no explicit guidance about self-adaptation quality attributes trade-offs was given during the lectures, since such an aspect is part of the hypotheses investigated herein.

In the 5th day, we conducted an one hour discussion about the matter, followed up by a three hours exam where

[3]http://hadoop.apache.org

students used pen and paper to answer open-ended questions. In the 6th day, we discussed the exam results and presented a 3 hours training session about the DuSE-MT tool and the architecture styles catalog for SA systems we developed for this experiment.

The experiment took place in the last two days of the course. Students were randomly assigned to two equal size groups, blocked by their stronger technical background (see subsection III-D). Since we undertook the experiment as a blocked subject-object study with three objects (web server initial model, MapReduce architecture initial model, and questionnaire) and two treatments (search-based approach and style-based approach), a total of twelve tests were undertaken (presented in Table II and discussed in subsection III-D). All design tests aimed at extending an initial model with a SA mechanism which regulates a performance metric, while yet minimizing the settling time, maximum overshoot, and control overhead. Both groups used DuSE-MT as the design tool, but all functionalities regarding design space navigation and architecture optimization were turned off when using the style-based approach as treatment. Conversely, students had no access to the style catalog when using the search-based approach. We would like to emphasize that the experiment was not intended to investigate design productivity and fault density, since those aspects are obviously favored when adopting automated design approaches.

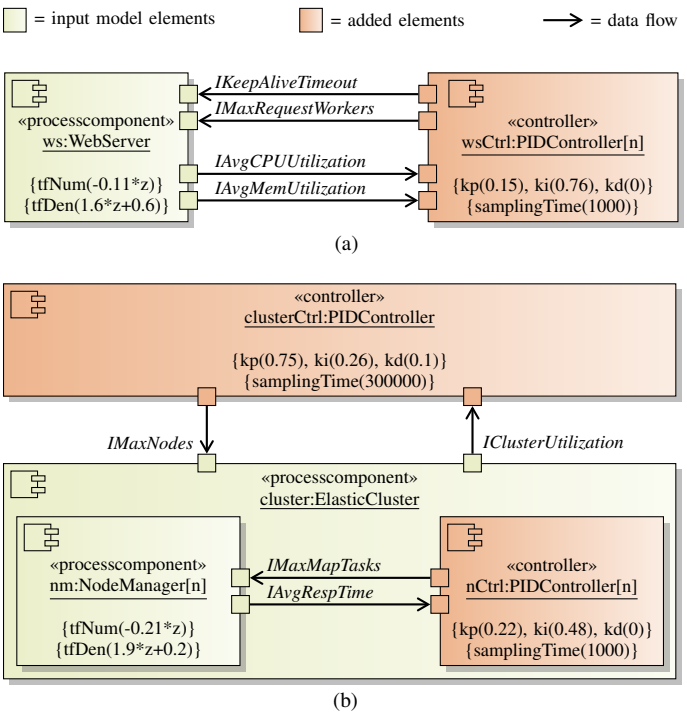

Fig. 2. Experiment objects (input model elements only): web server model (a) and MapReduce architecture model (b). The added elements exemplify the architectural extensions to be designed by the experiment subjects, either by applying the architectural styles catalog (reference approach) or by adopting our automated design space exploration mechanism (intervention approach).

A. Design Objects

The experiment's design tests aimed to create managing elements (adaptation loops) for two distinct managed elements: a web server and a MapReduce distributed architecture [12]. Such managed elements were used as experiment objects and are depicted in Fig. 2a and Fig. 2b as components with the "*input model elements*" key. Experiment subjects were expected to extend such input models with a particular feedback loop design that produces short settling times, minimum overshoot, and low control overhead. Fig. 2 shows two examples of such loops as components with the "*added elements*" key. We chose these experiment objects because they constitute two self-adaptation scenarios widely investigated nowadays and pose different design challenges: MIMO local control for the web server case study vs. SISO nested control in a distributed environment for the MapReduce architecture case study.

The web server model (WS) – depicted in Fig. 2a – entails a single component providing four interfaces: two for monitoring purposes (`IAvgCPUUtilization` and `IAvgMemUtilization`) and two for adjusting parameters that directly impacts the measured outputs (`IKeepAliveTimeout` and `IMaxRequestWorkers`). The goal is to retain web server's CPU and memory utilization as close as possible to the specified reference values, by simultaneously adjusting the number of threads serving HTTP requests (via `IMaxRequestWorkers` interface) and the amount of time the server will wait for subsequent requests on a given thread (via `IKeepAliveTimeout` interface).

The MapReduce architecture model (MR) – depicted in Fig. 2b – describes a distributed computing infrastructure (cluster) where an array of n nodes stores and analyzes huge datasets. The cluster infrastructure orchestrates the parallel execution of a Map function for each data block stored in cluster's nodes and combines all Map's outputs to form the Reduce function's input [12]. Apache Hadoop [21] is a well-established open source implementation of the MapReduce programming model, whose performance may be fine-tuned through nearly 190 configuration parameters. Although default values for such parameters are already provided by Hadoop, improvements of 50% in performance have been observed in properly configured setups [22]. In spite of that, Hadoop provides no services for parameter self-optimization or feedback control loops. The model we present in Fig. 2b entails two nested controllable components: `NodeManager` and `ElasticCluster`. Each cluster machine runs the `NodeManager` service, which may have its partial job's average response time (measured via `IAvgRespTime` interface) regulated by adjusting the maximum number of map tasks simultaneously executing in that host (Hadoop's `mapreduce.tasktracker.map.tasks.maximum` parameter, changed via `IMaxMapTasks` interface). Additionally, the overall cluster utilization (measured via `IClusterUtilization` interface) may also be regulated by adjusting the number of cluster hosts serving the job (via `IMaxNodes` interface).

B. Variables Selection

In this quasi-experiment, we are interested in analyzing the impact of the adopted design method on three dependent variables: the effectiveness of the resulting managing element, the complexity of managing element's architecture, and the method's potential for promoting the acquisition of insights

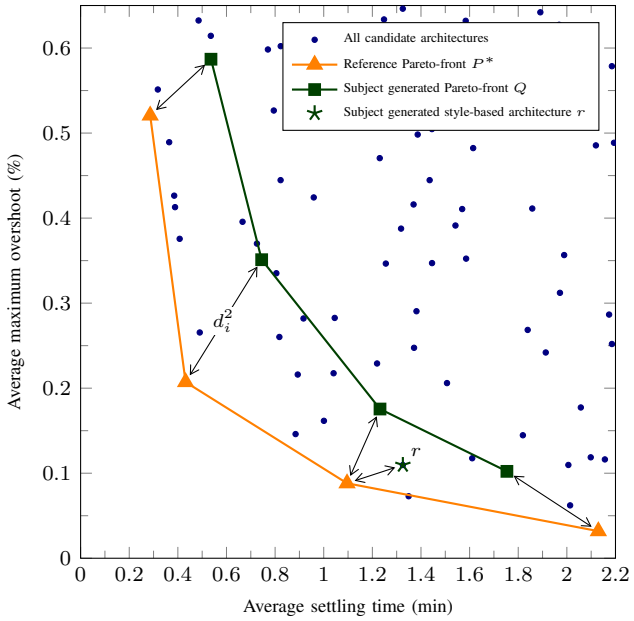

Fig. 3. Example of reference Pareto-front P^* and Generational Distance (GD) metric. GD finds the average Euclidean distance from each *i-th* solution in Q to the nearest solution in P^*. The lower the GD value, the more effective are the architectures regarding self-adaptation quality attributes. The style-based architecture r is shown as more effective than those produced by the search-based approach. That is one of the null hypotheses investigated herein.

and refined experience about quality attributes trade-offs involved in SA systems design. This subsection describes the metrics we adopted to quantify such variables.

1) Measuring Effectiveness (Generational Distance): we quantify the effectiveness of resulting feedback control loops in terms of how close their quality attributes are from a set of Pareto-optimal solutions previously obtained. Since meta-heuristics-based approaches – like ours – do not guarantee global optimality, we performed a set of 50 optimization runs and calculated the reference Pareto-front P^* of the union of all runs' outputs. A Pareto-front is a set of solutions for which it is impossible to make any other architecture better off without make at least another one worse. We assume that P^* (triangle path in Fig. 3) is a nice representative of the most effective solutions and constitutes a reasonable reference value for evaluating how effective are the architectures designed by the experiment subjects. We have done such procedure for both the objects (WS and MR) used in the experiment, producing the P^*_{WS} and P^*_{MR} reference Pareto-fronts.

The Generational Distance (GD) [14], [16] is a widely used metric to evaluate closeness between two Pareto-Fronts Q and P^*. The metric finds an average distance of the solutions of Q (or r) from P^*, as follows:

$$GD = \frac{(\sum_{i=1}^{|Q|} d_i^p)^{1/p}}{|Q|}; \text{ where } d_i^2 = \min_{k=1}^{|P*|} \sqrt{\sum_{m=1}^{M} (f_m^{(i)} - f*_m^{(k)})^2}$$

Any L^p-norm can be used in Generational Distance. For $p = 2$ as described above, d_i^2 is the Euclidean distance between the solution $i \in Q$ and the nearest member of P^*. We chose the Generational Distance because it is more suitable than alternatives like Error Ratio and Set Coverage [14] when comparing disjoint Pareto-fronts. Furthermore, its evaluation can be performed with minimal computational costs.

As for designs produced with the style-based approach (e.g., solution r in Fig. 3), their corresponding location in objective space (quality metrics values) were first calculated and then compared to the reference Pareto-front using Generational Distance. Note that a Pareto-front Q obtained with the search-based approach (e.g., square path in Fig. 3) is not necessarily as effective as the reference Pareto-front P^*, because of the inherent randomness in the adopted evolutionary multi-objective optimization approach (NSGA-II).

Although r is shown, in Fig. 3, more effective than any solution in Q, we believe that this is unlikely to happen in designs undertaken by architects with no previous experience in SA systems. Therefore, we look, in this experiment, for any evidence that supports/rejects our claim that search-based approaches may improve the effectiveness of such designs.

2) Measuring Complexity (Component Point): the second dependent variable we focus in this experiment is design complexity, since it directly impacts the development effort required to realize the proposed architectures. We used the Component Point (CP) approach [17] to quantify such an aspect, motivated by its original conception towards the measurement of UML models and by the existence of empirical evidence regarding its validity and usefulness [17]. CP provides the means to measure design complexity in terms of component's interfaces complexity and component's interaction complexity. We define the complexity CC_c for a component c as:

$$CC_c = IFCI_c + ITCI_c = \frac{IFC_c}{n_c} + \frac{ITC_c}{m_c}$$

$IFCI_c$ is the Interface Complexity per Interface, defined as the component's Interface Complexity (IFC_c) divided by the number of component's provided interfaces (n_c). Similarly, the Interaction Complexity per Interaction ($ITCI_c$) is defined as the component's Interaction Complexity (ITC_c) divided by the number of component's interactions (m_c). IFC_c and ITC_c, in their turn, are defined as follows.

The first step when calculating IFC_c is classifying each interface of a component into two types: ILF (Internal Logical Files) or EIF (External Interface Files). ILF interfaces are those whose operations change attributes of other interfaces, while the remaining interfaces as classified as EIF. The CP approach also specifies how a complexity level (Low, Average, High) should be assigned to each interface, based on the number of operations and number of operation's parameters it presents. Hence, IFC_c is defined as:

$$IFC_c = \sum_{j=1}^{2} \sum_{k=1}^{3} I_{jk} \times W_{jk}$$

I_{jk} is the number of interfaces of type j (1=ILF and 2=EIF) with complexity level k (1=Low, 2=Average, and 3=High). W_{jk} is the weight, given by the CP approach, for the interface type j with complexity level k.

ITC_c is evaluated in terms of the Interaction Frequency (IF_{ij}) of the *j-th* operation of the *i-th* interface and the Complexity Measure (CM_{ijk}) of the *k-th* data type involved in the execution of the *j-th* operation of the *i-th* interface.

TABLE II. TESTS DEFINED FOR THE EXPERIMENT.

#Test	Object	Treatment	Subjects
1	Web Server	Style-Based Approach	Group 1
2	MapReduce Architecture	Search-Based Approach	Group 2
3	MapReduce Architecture	Style-Based Approach	Group 1
4	Web Server	Search-Based Approach	Group 2
5	Questionnaire	Style-Based Approach	Group 1
6	Questionnaire	Search-Based Approach	Group 2
7	MapReduce Architecture	Search-Based Approach	Group 1
8	Web Server	Style-Based Approach	Group 2
9	Web Server	Search-Based Approach	Group 1
10	MapReduce Architecture	Style-Based Approach	Group 2
11	Questionnaire	Search-Based Approach	Group 1
12	Questionnaire	Style-Based Approach	Group 2

IF_{ij} is defined as a ratio of the number of interactions (N_O) performed by the operation and the number of interactions (N_I) performed by all operations of the interface. CM_{ijk}, in its turn, is defined as:

$$CM_{ijk}(D, L) = L + \sum_{n=1}^{m} CM(DT_n, L + 1)$$

D is the data type under measurement, L is the number of the level where the data type D occurs in the component data type graph (initially 1), DT_n is the data type of the *n-th* D's data member and m is the number of data members in D. Finally, ITC_c can be defined as:

$$ITC_c = \sum_{i=1}^{p} \sum_{j=1}^{q} \left(IF_{ij} \times \sum_{k=1}^{r} CM_{ijk} \right)$$

p is the number of interfaces provided by component c, q is the number of operations that the *i-th* interface provides, and r is the number of data types involved in the execution of the *j-th* operation of the *i-th* interface. The overall architecture complexity AC is defined as the sum of the CC_i's of every component i comprising the solution.

3) Measuring the Acquisition of Distilled Design Knowledge (Post-Experiment Questionnaire): the third dependent variable we investigated herein is the method's potential for leveraging the acquisition of distilled design knowledge. For that purpose, we prepared a questionnaire with 10 multiple choice questions related to refined knowledge about quality attribute trade-offs in the SA systems domain. Students answered such questionnaire at the end of each experiment day and we assigned grades according to the number of correctly answered questions. The goal was to evaluate to which extent the adopted design approach may leverage the acquisition of distilled knowledge about such design trade-offs. The questionnaire is available at the experiment website.

C. Hypotheses Formulation

In the quasi-experiment we report herein, we compare the use of a search-based architecture design approach and a style-based design approach with respect to the effectiveness and complexity of resulting architectures, as well as to the method's potential to promote the acquisition of distilled design knowledge. Such goal has been stated in three null hypotheses (H_0) and their corresponding alternative hypotheses (H_1):

- H_0^1: there is no difference in *design effectiveness* (measured in terms of the Generational Distance GD) between a feedback control loop design created using the style-based approach (reference approach: RA) and a feedback control loop design created using the search-based approach (intervention approach: IA).

$$H_0^1 \quad : \quad \mu_{GD_{RA}} = \mu_{GD_{IA}}$$
$$H_1^1 \quad : \quad \mu_{GD_{RA}} > \mu_{GD_{IA}}$$

- H_0^2: there is no difference in *design complexity* (measured in terms of the Architectural Complexity AC) between a feedback control loop design created using the style-based approach and a feedback control loop design created using the search-based approach.

$$H_0^2 \quad : \quad \mu_{AC_{RA}} = \mu_{AC_{IA}}$$
$$H_1^2 \quad : \quad \mu_{AC_{RA}} > \mu_{AC_{IA}}$$

- H_0^3: there is no difference in the *acquisition of distilled design knowledge* (measured in terms of applied questionnaire's grade QG) between designing a feedback control loop using the style-based approach and designing a feedback control loop using the search-based approach.

$$H_0^3 \quad : \quad \mu_{QG_{RA}} = \mu_{QG_{IA}}$$
$$H_1^3 \quad : \quad \mu_{QG_{RA}} < \mu_{QG_{IA}}$$

D. Experiment Design

The experiment was undertaken as a blocked subject-object study, which means that each subject exercises both treatments and effects can be compared in pairs. Since the experiment students had a stronger technical background in two different fields (14 devoted to software development and 10 devoted to network administration), we used such an information as a blocking factor. By doing that, we eliminate the undesired effect of student's technical background on the dependent variables, increasing the precision of the experiment.

Students from each technical background partition were randomly and equally assigned to experiment groups 1 and 2, yielding a similar proportion of developers and network administrators in each group. As presented in Table II, a total of eight design tests and four questionnaire answering tests were conducted in the experiment. Each group experienced every combination of an object (WS model, MR model, or the questionnaire) and a treatment (style-based approach or search-based approach). In the first experiment day, group 1 applied the style-based approach, initially in the WS model and then in the MR model, while group 2 adopted the search-based approach with the opposite object's order. At the end of the day, both groups answered the questionnaire based on their experience with the corresponding approach. In the second experiment day, groups exchanged the treatments and experienced the objects in the opposite order to the one conducted by them in the previous day. The same questionnaire was applied again at the end of the second day.

To minimize the effect of subjects gaining information from previous assignments, we systematically balanced which object-treatment combination is first experienced in each group. The tests' operation order is presented in Table I. To reduce hypotheses guessing and other social threats, students did not receive any feedback and were not aware of the experiment until its completion.

TABLE III. DESCRIPTIVE STATISTICS FOR THE EXPERIMENT'S DEPENDENT VARIABLES.

Generational Distance (GD)			
	mean(μ)	**median**	**std. dev.**
Search-Based Approach (IA)	2.40	2.45	1.08
Style-Based Approach (RA)	2.59	2.41	1.03
Difference (IA−RA)	-0.19	-0.62	1.32
Architecture Complexity (AC)			
	mean(μ)	**median**	**std. dev.**
Search-Based Approach (IA)	6.46	6.65	2.77
Style-Based Approach (RA)	7.02	7.05	2.70
Difference (IA−RA)	-0.57	-1.90	3.47
Questionnaire Grade (QG)			
	mean(μ)	**median**	**std. dev.**
Search-Based Approach (IA)	6.85	7.00	1.43
Style-Based Approach (RA)	7.04	7.25	1.27
Difference (IA−RA)	-0.19	-0.50	1.51

IV. ANALYSIS OF THE RESULTS AND DISCUSSION

After the experiment operation, 20 subjects provided usable data for paired comparison of Generational Distance and Architecture Complexity. Questionnaire answers were then restrict to those provided by such 20 subjects. With the support of DuSE-MT, all UML models resulting from the design tests were serialized in XML files, along with their corresponding quality attributes values (objective-space location). Such values were used to compute the Generational Distance for all resulting models. The Architecture Complexity value was also calculated for each resulting UML model.

A. Analysis

Figure 4 and Table III summarize the measured values of all dependent variables, as well as their paired difference with respect to the adopted treatment. The first step we undertook in the analysis stage was to investigate whether the usual assumptions for the use of parametric tests – preferable because of their enhanced power – hold in the collected data. Such assumptions are: *i)* data is taken from an interval or ratio scale (held for all experiment's dependent variables); *ii)* observations are independent (enforced by experiment design); *iii)* measured values are normally distributed in the populations; and *iv)* population variances are equal between groups (homoscedasticity).

We used the Anderson-Darling test [23] to evaluate to which extent the paired differences are normally distributed. The Brown–Forsythe test [24] was applied to investigate the null hypothesis of homoscedasticity between the intervention approach and reference approach groups. Table IV presents such a results. With a significance level (α) of 0.05, we observed that only the Questionnaire Grade (QG) paired difference could be considered normally distributed (Anderson-Darling p-value > 0.05). In addition, for all dependent variables, the null hypothesis of homoscedasticity could not be rejected (Brown-Forsythe p-value > 0.05). Since all assumptions must hold, only hypothesis H_0^3 was evaluated by a parametric test. The paired differences of Generational Distance (GD) and Architecture Complexity (AC) were not considered normally distributed (p-value $< \alpha = 0.05$) and, as such, hypotheses H_0^1 and H_0^2 were evaluated by using a non-parametric test.

TABLE IV. RESULTS OF ANDERSON-DARLING NORMALITY TEST AND BROWN–FORSYTHE HETEROSCEDASTICITY TEST ($\alpha = 0.05$).

Dependent Variable	**Anderson-Darling p-value**	**Brown–Forsythe p-value**
Generational Distance	1.29814505086953E-007	0.9009324909
Architecture Complexity	2.88672812219819E-010	0.7207666486
Questionnaire Grade	0.635529605	0.7167840476

As presented in Table V, we used the Wilcoxon Signed-Rank test [15], [25] to investigate H_0^1 and H_0^2 and the Paired t-test [15] to investigate H_0^3. With a significance level (α) of 0.05, H_0^1 and H_0^2 were rejected while no evidence could be found about H_0^3.

B. Discussion

The descriptive statistics and results of hypotheses tests show that there are improvements in the dependent variables, except for the Questionnaire Grade (QG). Actually, students who first exercised the search-based approach got slightly smaller grades (6.85) than other ones (7.04). Since both the architecture style catalog and the design space used in the experiment contain the same information, two possible reasons for such difference remain. First, students exposed to the search-based approach may had experienced a larger set of candidate architectures, which would contribute to obfuscate some quality trade-offs evaluated in the questionnaire. Second, the quality trade-offs may actually be not too difficult to grasp without the use of structured design spaces and automated architecture optimization, so that the difference is actually by chance. Further experiments are needed to better investigate such an aspect.

Generational Distance (adopted measure for effectiveness) is, on average, 7% lower with the search-based approach (2.40) when compared to the style-based approach (2.59). Architecture Complexity is, on average, 7% lower with the search-based approach (6.46) when compared to the style-based approach (7.02). While such values already indicate some improvements in the resulting architectures, we still lack further investigation about the boundaries that such enhancements may present.

V. THREATS TO VALIDITY

This section presents the threats to validity [15] we identified for the experiment reported in this paper.

A. Threats to Construct Validity

Construct Validity is the degree to which the objects and measurements reflect their associated constructs in the real world. We have identified three such threats: inadequate pre-operational explication of constructs, hypothesis guessing, and objects representativeness.

TABLE V. RESULTS OF STATISTICAL TESTS ($\alpha = 0.05$).

H_0^i	**Test**	**Criteria**	**Conclusion**
1	Wilcoxon Signed-Rank	T(410) > T-critical(378)	Rejected
2	Wilcoxon Signed-Rank	T(367) > T-critical(361)	Rejected
3	Paired t-test	p-value=0.5488018266	Not Rejected

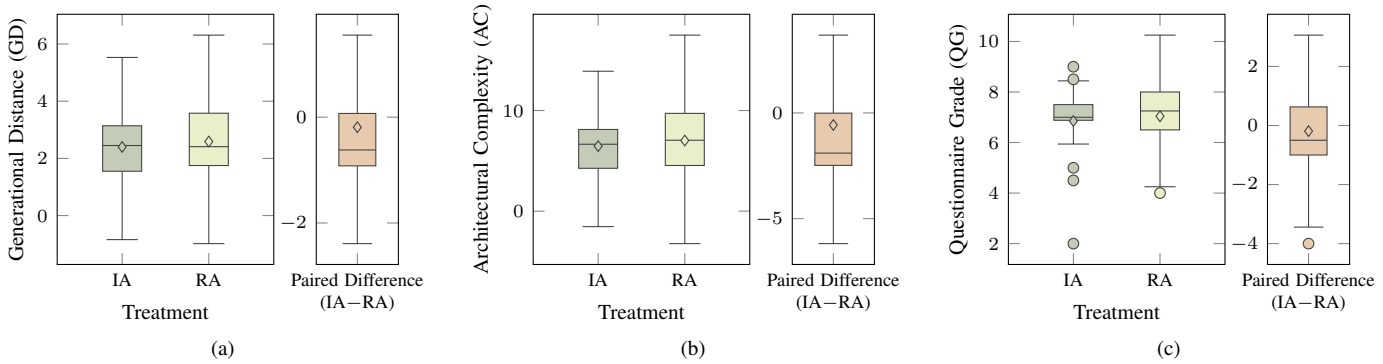

Fig. 4. Box and whiskers plots for design effectiveness (a), design complexity (b), and method's potential for leveraging the acquisition of distilled knowledge (c). For each dependent variable its shown the values for the search-based approach (IA: intervention approach), the style-based approach (RA: reference approach), as well as the values of the paired difference (IA−RA).

First, since the theory behind feedback control loops encompasses areas such as systems and signals, modeling of dynamic behavior, and analysis in frequency domain, students may have had no enough time to get a firm grasp about such mathematical background. To reduce this threat, we focused on requiring minimum knowledge about such as aspect and tried to leverage tool support regarding this issue in DuSE-MT. Second, students may have tried to perform better when using the search-based approach because it is the treatment proposed by the course holders. To mitigate this issue, students were not aware of the experiment and were graded on all tests. Third, the objects used in the experiment may not actually reflect the kind of problems routinely faced in the SA systems domain. Since the two adopted self-adaptation scenarios have been repeatedly investigated in a number of recent papers, we believe they constitute interesting and representative examples of current practice.

B. Threats to Internal Validity

Internal Validity concerns in analyzing to which extent unknown factors may affect the dependent variables with respect to causality. We have identified two such threats. The first one is maturation, where subject gain insights from previous experiment sessions. To reduce this threat, we alternately assigned such objects during the two experiment days. The second is related to instrumentation. Since a new modeling tool was adopted in the experiment (DuSE-MT), that may have impacted in some extent the student's abilities for developing the required models.

C. Threats to External Validity

External Validity is related to the ability of generalizing the experiment results to other settings. Since we used students of a graduate program in Distributed and Ubiquitous Computing, they may not represent the expected background in current industry practice.

D. Threats to Conclusion Validity

Conclusion Validity is related to the ability of generalizing the results to the overall concept or theory which supports the experiment. Since the experiment objects were created by us, there is a potential threat that such objects do not actually represent the problem under investigation. Such threat could have been reduced by relying on external SA systems experts to design such objects.

VI. Related Work

To the best of our knowledge, no controlled experiments regarding the use of search-based approaches when designing SA systems has been undertaken so far. However, we identified one experiment regarding SA systems design and a number of papers reporting on controlled experiments about software architecture design.

In [26], the authors report the results of a quasi-experiment that investigates whether the use of external feedback loops (when compared with internal adaptation mechanisms) improves the design of SA systems. The design was evaluated with respect to design complexity (in terms of activity complexity and control flow complexity), fault density, and design productivity. The experiment shows that external feedback loops reduce the number of adopted control flow primitives, increasing the design's understandability and maintainability. They also observed improvements in design productivity when using external feedback loops, but found no significant effects on design complexity in terms of activity complexity. The experiment we present herein tackles a similar design issue but with different treatments, objects, measurements, and hypotheses. While their experiment reveals evidence about the decoupling and reusability benefits of external feedback loops, we believe our experiment contributes by revealing the potential benefits of systematic design knowledge representation and search-based automated design approaches in such a domain.

A controlled experiment aimed at evaluating the impact of design rationale documentation techniques on effectiveness and efficiency of decision-making in the presence of requirements changing is presented in [27]. The results show that the use of such documentation technique significantly improves effectiveness but with no impacts on efficiency. In [28], the authors present a controlled experiment which evaluates the impact of the use of design rationale documentation on software evolution. They conclude that there are improvements in correctness and productivity when such documentation is available. Our search-based design approach under evaluation

herein supports rationale documentation in terms of domain-specific design spaces. The experiment we report in this paper is ultimately assessing the impact of having such rationale documented in a structured and systematic way, in contrast to ad-hoc styles catalogs or unstructured rationale documents.

In [29], a controlled experiment was performed to evaluate the usefulness of architectural patterns when evolving architectures to support specific usability concerns. The authors conclude that usability concerns are amenable to be handled in architectural level and that architectural patterns can significantly leverage such an aspect. In the SA systems domain, architecture-centric approaches with explicit (first-class) representation of feedback loops have been advocated as a promising research direction [19], [30], [31], due to their generality and support for early reasoning of self-adaptation quality attributes. The experiment we describe in this paper evaluates how search-based design approaches impact such first-class representation of feedback loops.

A controlled experiment on the impact of the use of design patterns on the productivity and correctness of software evolution activities in described in [32]. They conclude that each design pattern presents a specific impact on such dependent variables and, therefore, claim that design patterns should not be characterized as useful or harmful in general. In contrast, our experiment compares the use of two distinct representations of such distilled design knowledge: architectural styles vs. structured design spaces. Furthermore, we are interested in evaluating whether search-based design automation improves the effectiveness of SA systems.

With respect to software engineering mechanisms for SA systems, in [33] Weyns et al. present FORMS: an unifying reference model for formal specification of distributed SA systems. Their approach provides a small number of modeling elements capturing key design concerns in the SA systems domain. In contrast to our approach, FORMS provides no means for automated design of feedback loops and a steep learning curve may be experienced because of its rigorous formal underpinnings.

Vogel & Giese, in [34], propose a new modeling language for explicitly describing feedback control loops as runtime megamodels (multiple models@runtime). In contrast, our approach builds on top of widely accepted standards for modeling languages, like MOF and UML. Although our approach has been used as an off-line design mechanism, future work include moving such infrastructure to runtime, providing a models@runtime approach for Dynamic Adaptive Search-Based Software Engineering [35].

A UML profile for modeling feedback control loops as first-class entities is presented in [30]. In our mechanism, we go a step further towards the use of UML profiles as the underlying mechanism for identifying *loci* of architectural decisions, enabling automated design, and detecting invalid candidate architectures.

In [36], Cheng et al. present an adaptation language which relies on utility theory for handling self-adaptation in the presence of multiple objectives. *A priori* preference articulation methods – like utility functions – convert a multi-objective optimization problem into a single-objective one, but its effectiveness highly depends on an well-chosen preference vector. Our approach, on the other hand, accommodates the multi-objective nature of SA systems design as an essential aspect by using *a posteriori* preference articulation.

An on-line learning-based approach for handling unanticipated changes at runtime is presented in [37]. Whilst we have considered in this work only feedback control as the enabling mechanism for self-adaptation, other strategies may be modeled as new variation points.

In [38], Křikava et al. propose a models@runtime approach which represents adaptation logic as networks of messaging passing actors. Our work, in contrast, leverages design reuse by requiring the use of highly distilled design knowledge only once – when designing a domain-specific DuSE design space. Thereafter, novice architects have better support for designing effective architectures and getting insights from the search activities.

VII. Conclusions

This paper presented a quasi-experiment aimed at evaluating whether search-based architecture design approaches improve the effectiveness and complexity of SA systems when compared to style-based design approaches. To the best of our knowledge, that is the first endeavor in evaluating how search-based automated design impacts the quality of SA systems. The results reveal that the use of systematically structured design spaces and architecture optimization mechanisms indeed provide enhanced support to the evaluation of quality trade-offs, for the experiment objects considered herein.

Some insights have been identified from the experiment results. We found no evidence that search-based approaches leverage the acquisition of distilled design knowledge in the SA systems domain. Alternative instruments for evaluating such an aspect may be adopted in future research, enabling the eliciting of more elucidative conclusions. However, search-based design approaches do contribute in revealing architectures which indeed exhibit a near-optimal trade-off between quality attributes. In contrast, architects using the style-based approach are more likely to design sub-optimal architectures. Improved effectiveness results in managing elements with lower overhead and enhanced use of resources, leveraging the overall SA behavior. Moreover, designs with lower complexity were also obtained when using the search-based approach, fostered by the systematic representation of the architecture changes required to realize the involved feedback loops. As a consequence, one should expect positive effects in understandability, maintainability, and testability of development artifacts realizing such architectures.

A lot of current research are driving their efforts towards the establishment of principled and well-founded underpinnings for engineering software-intensive systems, specially in particular application domains like SA systems. The organization of software design knowledge for routine use is mandatory if we are to realize the upcoming generation of software-intensive systems.

References

[1] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, june 2012, pp. 33 –42.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 14:1–14:42, May 2009.

[3] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, pp. 7:1–7:28, Aug. 2008.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[5] R. de Lemos et al, "Software Engineering for Self-Adaptive Systems: A second Research Roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Dagstuhl Seminar Proceedings, R. de Lemos, H. Giese, H. Müller, and M. Shaw, Eds., no. 10431. Dagstuhl, Germany.: Schloss Dagstuhl. Leibniz-Zentrum fuer Informatik, Germany. Full citation: http://dx.doi.org/10.1007/978-3-642-02161-9_1, 2011.

[6] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2379776.2379787

[7] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," in *FASE*, ser. Lecture Notes in Computer Science, D. S. Rosenblum and G. Taentzer, Eds., vol. 6013. Springer, 2010, pp. 1–12.

[8] O. RäIhä, "Survey: A survey on search-based software design," *Computer Science Reviews*, vol. 4, no. 4, pp. 203–249, Nov. 2010.

[9] S. S. Andrade and R. J. d. A. Macêdo, "Searching for effective feedback control architectures for distributed self-adaptive systems," *Technical Report*, Apr. 2014. [Online]. Available: http://dusearchitects.wordpress.com/publications/

[10] S. S. Andrade and R. J. d. A. Macêdo, "A search-based approach for architectural design of feedback control concerns in self-adaptive systems," in *Proceedings of the 7th IEEE Intl Conf. on Self-Adaptive and Self-Organizing Systems*. Philadelphia, PA, USA: IEEE, 2013.

[11] S. S. Andrade and R. J. d. A. Macêdo, "Architectural design spaces for feedback control concerns in self-adaptive systems," in *Proceedings of the 25th Intl Conf. on Software Engineering and Knowledge Engineering*. New York, USA: ACM, 2013.

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[14] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.

[16] D. A. Van Veldhuizen and G. B. Lamont, "On measuring multiobjective evolutionary algorithm performance," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 1. IEEE, 2000, pp. 204–211.

[17] T. Wijayasiriwardhane and R. Lai, "Component Point: A system-level size measure for component-based software systems," *J. Syst. Softw.*, vol. 83, no. 12, pp. 2456–2470, Dec. 2010.

[18] J. Andersson, R. Lemos, S. Malek, and D. Weyns, "Software engineering for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems*, B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Modeling Dimensions of Self-Adaptive Software Systems, pp. 27–47.

[19] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Software engineering for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems*, B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_3

[20] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[21] (2014) Apache Hadoop - The Apache Software Foundation. [Online]. Available: http://hadoop.apache.org

[22] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The performance of MapReduce: An in-depth study," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 472–483, 2010.

[23] G. Corder and D. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2009. [Online]. Available: http://books.google.com.br/books?id=-ufOfzVp6qYC

[24] P. I. Good, *Permutation, parametric and bootstrap tests of hypotheses*. Springer, 2005, vol. 3.

[25] J. D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference, Fourth Edition: Revised and Expanded*, ser. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 2003. [Online]. Available: http://books.google.com.br/books?id=kJbVO2G6VicC

[26] D. Weyns, M. U. Iftikhar, and J. Söderlund, "Do external feedback loops improve the design of self-adaptive systems ? a controlled experiment," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2013, pp. 3–12.

[27] D. Falessi, G. Cantone, and M. Becker, "Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 134–143.

[28] L. Bratthall, E. Johansson, and B. Regnell, "Is a design rationale vital when predicting change impact ? a controlled experiment on software architecture evolution," in *Product Focused Software Process Improvement*. Springer, 2000, pp. 126–139.

[29] E. Golden, B. E. John, and L. Bass, "The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 460–469.

[30] R. Hebig, H. Giese, and B. Becker, "Making control loops explicit when architecting self-adaptive systems," in *Proceedings of the 2nd International Workshop on Self-Organizing Architectures*, ser. SOAR '10. New York, NY, USA: ACM, 2010, pp. 21–28. [Online]. Available: http://doi.acm.org/10.1145/1809036.1809042

[31] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, ser. ULSSIS '08. New York, NY, USA: ACM, 2008, pp. 23–26. [Online]. Available: http://doi.acm.org/10.1145/1370700.1370707

[32] M. Vokáč, W. Tichy, D. I. Sjøberg, E. Arisholm, and M. Aldrin, "A controlled experiment comparing the maintainability of programs designed with and without design patterns – a replication in a real programming environment," *Empirical Software Engineering*, vol. 9, no. 3, pp. 149–195, 2004.

[33] D. Weyns, S. Malek, and J. Andersson, "FORMS: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, p. 8, 2012.

[34] T. Vogel and H. Giese, "A language for feedback loops in self-adaptive systems," in *Proc. of the 7th Intl Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Computer Society, 6 2012, pp. 129–138.

[35] M. Harman, E. K. Burke, J. A. Clark, and X. Yao, "Dynamic adaptive search based software engineering," in *ESEM*, P. Runeson, M. Höst, E. Mendes, A. A. Andrews, and R. Harrison, Eds. ACM, 2012, pp. 1–8.

[36] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems*, ser. SEAMS '06. New York, NY, USA: ACM, 2006, pp. 2–8. [Online]. Available: http://doi.acm.org/10.1145/1137677.1137679

[37] N. Esfahani, A. Elkhodary, and S. Malek, "A learning-based framework for engineering feature-oriented self-adaptive software systems," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1467–1493, 2013.

[38] F. Křikava, P. Collet, and R. B. France, "Actor-based runtime model of adaptable feedback control loops," in *Proceedings of the 7th Workshop on Models@run.time*, ser. MRT '12. New York, NY, USA: ACM, 2012, pp. 39–44. [Online]. Available: http://doi.acm.org/10.1145/2422518.2422525