eQual: Search-Based Software Design Optimization

Anonymous Author(s)

ABSTRACT

When designing a software system, architects make a series of design decisions that directly impact the system's quality. Recent studies have shown that the number of available design alternatives grows rapidly with system size, creating an enormous space of intertwined design concerns. This paper presents *eQual*, a novel model-driven technique for simulation-based assessment of architectural designs that helps architects understand and explore the effects of their decisions. We demonstrate that *eQual* effectively explores massive spaces of design alternatives and significantly outperforms state-of-the-art approaches, without being cumbersome for architects to use.

1 INTRODUCTION

At the outset of a software development project, a set of critical design decisions that form the system's architecture needs to be established. Those decisions may be made anew, reused from previous systems' designs, or guided by chosen architectural styles and patterns or implementation frameworks and libraries. Architectural design decisions span a wide range of issues, from system structure, to behavior, interaction, deployment, evolution, and a variety of non-functional properties [68]. It has been long accepted that the *number* of design decisions grows quickly with the complexity of the system. A recent study corroborates this observation by proposing and empirically evaluating a technique for estimating that number for existing systems [63]. The difficulty of designing software systems lies not only in the sheer number of design decisions, but also in the fact that many intertwined factors and *trade-offs* must be taken into consideration [12, 13, 56].

Every architectural design decision is made by selecting one of several possible *alternatives* for a given *variation point* in a software system. For example, engineers may select between 1 and 5 authentication servers for their system. In this example, the authentication servers comprise the variation point, while choosing a specific number of servers is the selection of the alternative. An architecture *variant* is the set of design decisions whose outcome is the selection of an alternative for each variation point in a system, i.e., a *complete architecture* for the system. In our example system, in addition to the authentication server variation point, an architectural variant would include the selection of concrete alternatives for many other variation points, such as the number of replicated customer data stores, the choice of each data store (e.g., different candidate implementations of NoSQL databases and the relevant specific parameters of each), the choice of implementation framework (e.g., different MVC-based Web frameworks and their relevant parameters), overarching architectural style(s) and their design implications (e.g., layered client-server that may yield an N-tiered architecture vs. event-based that may result in distributed peers), different candidate data caching and pre-fetching strategies to reduce interaction latencies, and so on.

Ideally, individual alternatives would be carefully assessed to make viable design decisions that satisfy a system's requirements. However, this is frequently not done in practice [16]. A well publicized recent example is the Healthcare.gov portal (a.k.a. Obamacare) that was marred with serious technical problems at launch [42, 52, 71]. Several studies have pointed out that the failures causing downtimes of up to 60% were due to flawed architectural and deployment design decisions [72, 73]. As a result, Obamacare's development costs, originally estimated to be ~$100M, surpassed $1.5B [40].

The root-cause of such project failures is actually known: evaluating system design options is an exceedingly difficult and complex problem. The space of system variants grows exponentially in the numbers of design decisions and alternatives. Thus, manually comparing the variants that are available to architects is beyond human capabilities for most systems [28]. Prior work has aimed to support engineers in this decision-making and to help them evaluate and compare variants. Such comparisons require objective assessments of each variant [35, 56]. The existing approaches rely on static or dynamic analysis for assessing software models that represent the variants. Static analysis techniques (e.g., [7, 27, 47]) tend to require architects to develop complex mathematical models, which imposes steep learning curves, significant modeling effort, and limits on the resulting system's scalability. Additionally, depending on the mathematical models they rely on (e.g., Markov chains [26], event calculi [38], or queueing networks [8]) these techniques are confined to specific kinds of software system models [2].

While they come with shortcomings of their own (e.g., false negatives, longer execution times), dynamic analysis techniques — i.e., architectural model *simulations* [22, 43] — are more capable of capturing the nondeterminism reflective of reality [36] and are more amenable to constructing models that are tailored to the task at hand. Despite notable efforts [18, 45, 70, 74], simulations of software architectural models have not been as widely employed as traditional static analyses [2], due to at least four reasons. First, creating simulatable system design models is acknowledged as *difficult* [22]. Second, running simulations on complex models is time consuming, mandating that *scalability* issues be explicitly addressed [55]. Third, quantitative assessment of variants is a *complex* computational problem because of the large number of involved trade-offs [48]. Finally, depending on the size of the simulations, *massive datasets* may need to be analyzed and understood to assess system behavior.

To address the shortcomings of existing approaches, we have developed *eQual*, a model-driven architecture-based technique that enables simulation-driven assessment of architectural designs and design variants. *eQual*'s goals are three-fold

(1) ease of use,
(2) effectiveness in producing accurate solutions, and
(3) scalability to large models.

eQual enables engineers to explore a potentially massive space of design decisions while minimizing the burden on the engineers. *eQual* does so by asking a small number of relatively simple questions about the system the engineers are designing and their preferences for the system. In response, *eQual* automatically builds the requisite system models, runs simulations in the background, and delivers

to the engineers the ranked list of variants that correspond to their current design choices. The engineers can then adjust their preferences, tighten or relax the acceptable bounds on system parameters, or explore other variants, to reach the most appropriate architecture for the problem at hand.

eQual takes two inputs: (1) a model of the system and (2) answers to a set of questions. An example question is, "What is the maximum allowed value for this variation point?" *eQual* provides interactive facilities for creating the system model in a domain-specific language (DSL), although software architects already frequently create such models as part of their normal design processes [68]. The model captures the system's elements (e.g., components and connectors), their behavioral descriptions, and their overall composition into the system's configuration. *eQual*'s questions (1) bound the search space of system variants *eQual* will explore, and (2) identify the non-functional properties that are of interest, thus improving *eQual*'s efficiency and the quality of the results. In the extreme, we do not assume that architects will be able to answer any of the questions, so that *eQual* may have to explore an unbounded search space. *eQual* uses the system model and any specified bounds to generate and simulate system variants, eliminates the least desirable variants based on the of-interest properties, and iterates until it arrives at a set of satisfactory variants.

We evaluate *eQual* in three ways, corresponding to its three goals stated above. First, we demonstrate that using *eQual* is significantly easier than the state-of-the-art alternative that targets the same problem, GuideArch [24]. As a representative illustration, an architect only needed to spend six minutes to interactively answer the 27 questions *eQual* requires to analyze a recently published model of Hadoop [9, 10], while GuideArch's 356 inputs required more than four hours. Second, we extensively evaluate *eQual*'s effectiveness. We show that the quality of the designs *eQual* produces is higher than that of GuideArch. We additionally evaluate the quality of designs yielded by *eQual* against previously published ground-truths obtained from real-world software systems. We show that *eQual* recommends variants that are of comparable quality to the real-world solutions determined to be optimal, while it significantly outperforms the nominal solutions that are most commonly selected by architects. Third, we demonstrate that *eQual* scales optimally with the number of available computational nodes, system events, and system variants. In our experiments, *eQual* was able to analyze tens of thousands of variants and terabytes of simulation-generated data.

This paper's primary contributions include:

(1) A method for automatically generating architectural assessment models from simple inputs that architects provide.
(2) Bipartite Relative Time-series Assessment, a technique for efficient, distributed analysis of simulation-generated data, solving a previously prohibitively inefficient variant-assessment problem.
(3) An architecture for seamlessly distributing and parallelizing simulations to multiple nodes.
(4) An evaluation of each of *eQual*'s three goals on real-world systems, comparing to the state-of-the-art alternative, and demonstrating its ease of use, accuracy of produced solutions, and scalability.

This work makes an added contribution that uses prior research but has significant practical utility for the problem at hand:

(5) An extensible platform for using general-purpose or proprietary evolutionary algorithms to automate design-space exploration.

eQual is open-source. We will make it public in an unblinded version of this paper, along with all evaluation data and scripts.

The rest of our paper is structured as follows. Section 2 introduces an example of Hadoop, a real-world system we use to help describe evaluate *eQual*. Section 3 describes our approach and Section 4 evaluates it. Section 5 places our research in the context of related work and Section 6 overviews future research directions and summarizes our contributions.

2 RUNNING EXAMPLE

We will use a model of Hadoop to describe (Section 3) and evaluate (Section 4) *eQual*. In this model, a *computation* is the problem being solved. A *task* is an operation that is performed on the input (e.g., `map` or `reduce` in Hadoop). Tasks can be replicated, e.g., for reliability [9, 10]. A *job* is one instance of a task. *Machines* can execute these jobs. A *task scheduler* breaks up a computation into tasks, creates jobs as copies of these tasks, and assigns jobs to machines in the machine pool. After returning a response to the task scheduler, each machine rejoins the pool and can be selected for a new task. Figure 1 depicts a partial system model captured in a DSL (described in Section 3.1).

Although *eQual* has been used to analyze Hadoop's entire design space, for simplicity of exposition, our description and evaluation in this paper will focus on Hadoop's *variation points* that affect its key non-functional properties. There are five such variation points: (1) *Computation Size*, the number of tasks required to complete a computation; (2) *Redundancy Level*, the number of machines that will run identical jobs; (3) *Pool Size*, the number of available machines; (4) *Machine Reliability*, the probability of a machine returning the correct result before a timeout; and (5) *Processing Power* of each machine. Figure 2 depicts representative bounds for the

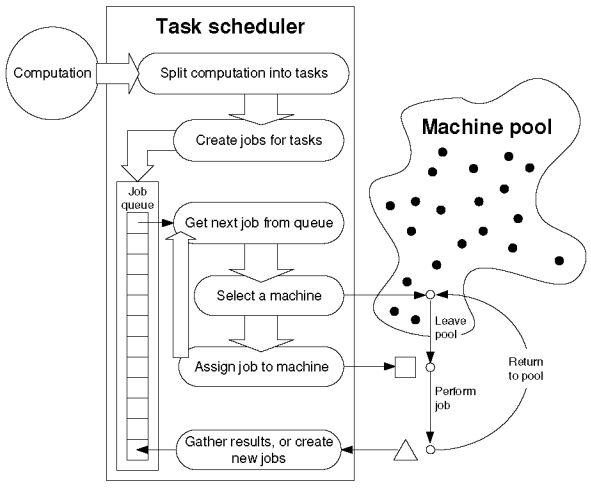

Figure 1: Hadoop system model in a domain-specific language [9].

Variation Point	Lower Bound	Upper Bound
Computation Size	500	2000
Redundancy Level	1	5
Pool Size	10	100
Machine Reliability	0.5	0.9
Processing Power	0.5	2

Figure 2: Hadoop variation points and representative bounds [9, 10].

variation points. This choice of variation-point bounds does not in any way impact *eQual*, its analysis, or its comparison to competing approaches. We have opted for these bounds simply because they were obtained from a previously published analysis of a volunteer computing platform that shares a number of characteristics with Hadoop [9, 10].

3 THE *eQual* APPROACH

eQual helps engineers explore the design space for their system via four steps: (1) modeling, (2) preparation, (3) selection, and (4) assessment. Steps (1) and (2) are interactive and generate *eQual*'s inputs: a system's design model and the answers to a set of design-related questions. *eQual* uses steps (3) and (4) to produce a list of ranked system variants. Steps (1) and (3) adapt existing solutions, while (2) and (4) are novel contributions of *eQual*. Steps (3) and (4) take place iteratively: the outputs of assessment feed back into selection, helping *eQual* to choose better alternatives, thereby generating improved system variants.

Critically, *eQual*'s required inputs are types of artifacts the architects should already have created or have the knowledge to create reasonably easily. Modern software development typically involves modeling of the system's architecture (*eQual*'s step 1), although sometimes informally or even implicitly [68]. Likewise, the answers to questions *eQual* asks (*eQual*'s step 2), which result in specifications of the desired behavioral properties in a system, are concerns the architects have to consider regardless of whether they use *eQual*.

eQual's overall architecture, with the components performing the four steps clearly denoted, is depicted in Figure 3. The remainder of this section details each of *eQual*'s four steps.

3.1 Modeling

eQual's first input is a system's architectural model that is amenable to dynamic analysis. Several existing approaches enable creating such models, including ArchStudio [17], XTEAM [22], PCM [46], and DomainPro [62]. Each of these as well as other similar approaches would suit our purpose. For *eQual*'s implementation we selected DomainPro [20, 62] because of its simple interface, dynamic analysis capabilities that use event-driven simulation, and its model-driven architecture (MDA) approach [51] that allows architects to define variation points in their models and try different alternatives (although completely manually).

As is common with MDA approaches, a system is designed in DomainPro in two phases. First, an engineer must specify or reuse a previously defined metamodel for the system. Second, the engineer designs the system by specializing and instantiating elements of this metamodel. A metamodel is a collection of design building blocks (e.g., components, interfaces, ports, services, hosts, resources, etc.) that are relevant to modeling systems of certain kinds or in certain domains. For example, Android-based systems are most naturally modeled using concepts such as activities, fragments, and content providers, while certain Web-based systems are best modeled using services.

DomainPro provides a built-in, generic metamodel for component-based architectures [49]. We employ this metamodel in *eQual*'s design, implementation, and evaluation. While more targeted metamodels for different classes of systems (e.g., highly distributed, Web-based, mobile, IoT, etc.) are likely to increase the effectiveness of *eQual*'s subsequent steps, this would result in additional burden on architects and would render more difficult direct comparison to techniques, such as GuideArch, that are not MDA-based.

The Hadoop model in Figure 1 uses DomainPro's generic metamodel by specializing and instantiating the metamodel elements [62]. Figure 1 represents Hadoop's model using the resulting visual DSL: a *Computation* is a DomainPro `Operation` depicted as a circle; each activity in the *Task Scheduler* `Component` is a DomainPro `Task` depicted as an oval; *Machine Pool* is a DomainPro `Resource` depicted as the shape containing the filled-in circles; data-flows are DomainPro `Links` represented with wide arrows; and so on. Figure 1 omits the DomainPro `Parameters` for each modeling element for clarity; Figure 2 lists the key parameters we reference in this paper.

3.2 Preparation

eQual's second input consists of answers to a set of questions that fall into two categories: the system's (1) variation points and (2) properties of interest (e.g., scalability and dependability). *eQual* formulates these questions in terms of the system's design, presenting specific choices that are intended to be straightforward for architects.

A. Questions Regarding Variation Points

For each variation point V, *eQual* asks architects three questions:

(1) What is V's lower bound?
(2) What is V's upper bound?
(3) What is the desired function for exploring V?

The lower and upper bounds capture the acceptable range of alternatives for each variation point. Exploration functions enable architects to customize how *eQual* samples the specified range in the process of design exploration (as detailed in Sections 3.3 and 3.4). For example, in our model of Hadoop, the *Pool Size* variation point has the lower bound of 10 and upper bound of 100 (recall Figure 2). The current prototype of *eQual* provides 12 exploration functions: Uniform, Poisson, Gamma, Exponential, etc. *eQual* also allows architects to provide lists of concrete values instead of ranges.

As shown by a prior analysis [63], well over 100 design decisions have been made during the development of Hadoop. Each design decision involved selecting an alternative for a variation point. Hadoop's architects had considered as few as 2 and as many as 8 alternatives per variation point. Even the 68 minor Hadoop revisions analyzed in [63] introduced up to 4 new design decisions each. Exploring the effects of just those newly introduced design decisions could be non-trivial. While manually exploring the resulting system variants by considering 1-2 alternatives per decision might be feasible, the resulting decision space grows exponentially with

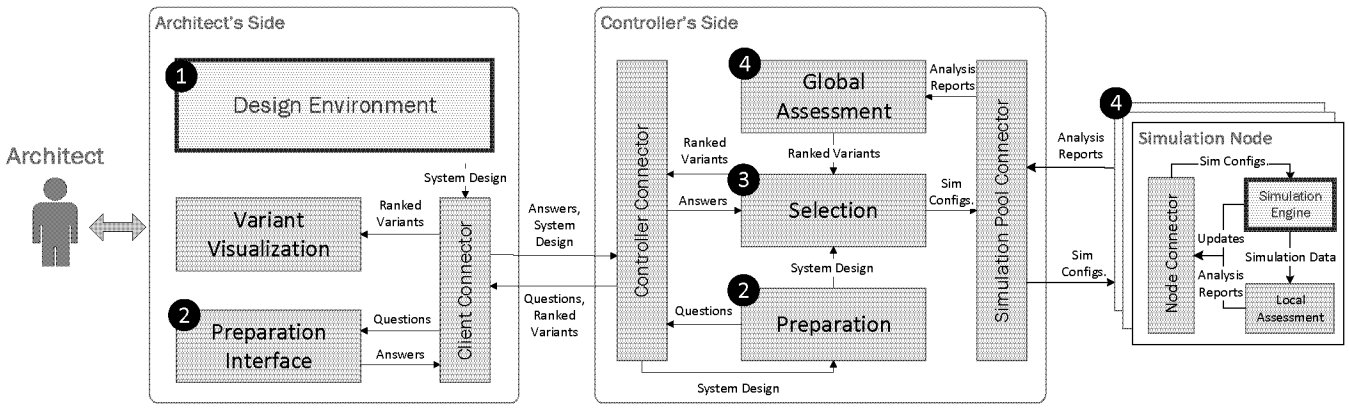

Figure 3: *eQual*'s architecture. The *Design Environment* and *Simulation Engine* components are provided by DomainPro [62].

the number of alternatives and quickly eclipses human abilities. For example, a Hadoop version that involves merely 4 design decisions and 5 alternatives per decision will have more than 500 variants. By contrast, the burden *eQual* places on architects is to answer $4 \times 3 = 12$ questions about the involved variation points.

We do not expect architects to be able to answer the above questions right away. For example, an architect may not be sure what the lower bound for the size of the node pool or the upper bound for the redundancy level should be. Instead, *eQual* allows several possibilities: (1) Architects may know the exact answer to a question, e.g., based on the requirements or domain knowledge. (2) Architects may be able to provide a partial answer, such as a property's lower bound. (3) Architects may be unable to answer the questions, leaving the range of alternatives unbounded. This allows architects to explore properties for which they have different degrees of knowledge.

B. Questions Regarding Non-Functional Properties

eQual's second set of questions deals with the system's desired non-functional properties. These properties are the basis for assessing design alternatives (as discussed in Section 3.4). The non-functional properties are determined from the system requirements and the characteristics of the domain. For example, in Hadoop, prior work identified four properties [9, 10]: (1) *Reliability* (ratio of tasks that have to be restarted to all tasks), (2) *Machine Utilization*, (3) *Execution Time*, and (4) *Cost* (the total number of executed jobs). Each property has to be tied to an aspect of the output of the system's dynamic analysis. In DomainPro and other approaches that use discrete-event simulation for dynamic analysis of system models (e.g., IBM's Rational Rhapsody [33]), the state of a system is captured at the times of event occurrences. Hence, the output is a set of time-series objects regarding different aspects of the simulated system.

For each non-functional property P, *eQual* asks three questions:

(1) What time-series object is of interest?
(2) Is P directly or inversely related to overall system quality?
(3) What is P's importance coefficient?

For example, in the case of Hadoop's *Machine Utilization* property, the relevant time-series object is the one capturing the idle capacity of the machine pool in the Hadoop model (recall Figure 1); the direction of the relationship is inverse (lower idle capacity means higher utilization); and the importance coefficient may be set to 3 (an ordinal value that would treat *Machine Utilization* as more important than, e.g., *Cost* whose coefficient is 1, and less important than, e.g., *Reliability* whose coefficient is 5). Thus, for the above-discussed example of a newly introduced Hadoop minor version with 4 variation points, given Hadoop's 4 properties of interest, an architect using *eQual* would have to answer a total of 24 questions: 12 questions for the variation points, and 12 for the properties.

eQual's objective is to *elaborate the information architects already must take into account*, without further burdening them. In current practice, architects will often ignore, accidentally omit, indirectly consider, or incorrectly record information captured by these questions. The many well-known software project failures provide ample evidence of this [68]. By consolidating the questions into one place and a standard format, *eQual* aims to convert this frequently haphazard process into a methodical design. As the architects explore the design alternatives and gain a better understanding of the system, they will be able to go back and add, remove, or change their answers.

3.3 Selection

The system's design model (Section 3.1) and the answers pertaining to the system's variation points and properties (Section 3.2) are the inputs to the selection step, whose objective is to explore the space of design variants intelligently and to make it tractable. For example, in Hadoop, this can help the engineer explore the effects of non-trivial decisions, such as: What yields better system reliability at an acceptable cost, a greater number of less-reliable machines or fewer more-reliable machines? How will this choice affect system performance?

Selection begins by generating an initial set of variants, i.e., by making an initial selection of alternatives for the system variation points using the information provided by the architects during preparation (Section 3.2). We call this initial set *seed variants* and process *eQual* uses to pick the seed (and later) variants the *selection strategy*. The seed variants feed into assessment (Section 3.4), where

eQual comparatively analyzes the variants. Assessment feeds its ranking of the variants back to selection (recall Figure 3), which in turn uses this information to generate an improved set of variants during the next iteration.

The key factors that determine the effectiveness of the selection process are the manners in which (1) the seed variants are generated and (2) the information supplied by the assessment step is used to generate subsequent variants. In principle, *eQual* allows any such selection strategy. The objective behind allowing different strategies is to allow an architect to control the selection step's number of iterations and generated variants to fit her needs, specific context, and available computational resources.

In our prototype implementation, we have explored two selection strategies based on the genetic evolutionary-algorithm paradigm: random seeding and edge-case seeding. In random seeding, we choose the seed variants completely randomly. In edge-case seeding, we aim to generate as many variants as possible containing either side of the boundary conditions that have been provided to *eQual*. For example, in Hadoop one variant would be generated by selecting all lower-bound values from Figure 2 (500, 1, 10, 0.5, 0.5), another by selecting all upper-bound values (2000, 5, 100, 0.9, 2), a third by selecting upper-bound values for the top three variation points and lower-bound values for the remaining two variation points (2000, 5, 100, 0.5, 0.5), and so on. Note that edge-case seeding is not possible in cases where options are nominal and do not have binary or numerical values assigned to them.

Both strategies are able to quickly prune the space of variants and arrive at good candidate designs. However, our empirical evaluation (discussed in Section 4) indicates that the edge-case heuristic is more efficient and arrives at better solutions. We aim to preserve Pareto optimal [11] solutions at each step since this provides an intuitive way to explore the extreme effects of decisions, giving architects insight into the different system aspects and their inter-relationships.

3.4 Assessment

To assess a variant's quality, *eQual* dynamically analyzes it via simulation. We have opted for simulation-based analysis because simulations are representative of a system's real behavior due to their inherent nondeterminism [36]. *eQual* specifically relies on discrete-event simulations, generating outputs in the form of time-series objects. Comparing different variants thus requires an analysis of their simulation-generated time-series. Although there are dozens of similarity metrics, in most domains (e.g., robotics, speech recognition, software engineering) Dynamic Time Warping (DTW) has been shown to perform better than the alternatives [19]. We thus use DTW.

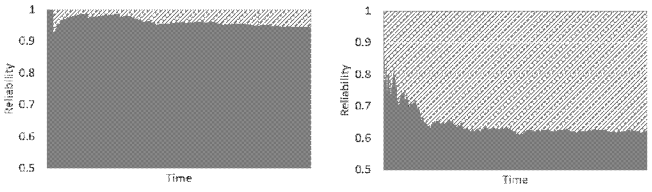

Figure 4: Reliability time-series of two Hadoop variants. The left variant uses machines with 90% reliability and no redundancy; the right variant uses machines with 50% reliability and redundancy factor of 2.

For each design variant, *eQual* generates a single time-series object for each non-functional property. For Hadoop, that means four time-series per variant, corresponding to the system's (1) *Reliability*, (2) *Execution Time*, (3) *Machine Utilization*, and (4) *Cost*. Each data point in a time-series corresponds to the computed value for the given property at the given time.

Depending on the direction of the relationship of a property with the overall system quality, we aim to find the variant that has yielded the time-series with the highest (direct relationship, e.g., for *Reliability*) or lowest (inverse relationship, e.g., for *Cost*) values for that property. To this end, we need to compare each time-series with the *optimum time-series*. The optimum time-series for a given non-functional property is a constant time-series each of whose data points is equal to the highest (or lowest) value of the property achieved across all simulations. This comparison requires having access to all of the simulation-generated data in one place, and computing the global optimum and distances from it. In turn, this may entail transferring hundreds of megabytes of data per variant, and having to redo all of the calculations in case a new variant is added that changes the optimum time-series. Such a solution would cause prohibitive performance and scalability problems in scenarios with multiple iterations involving thousands of variants.

To address this problem, we devised the *Bipartite Relative Time-series Assessment* technique (BRTA). BRTA enables distribution of the time-series analysis and eliminates the need to transfer all simulation-generated data to a central node. Instead, as indicated in Figure 3, multiple nodes are tasked with assessing different subsets of variants via simulation; a node may be responsible for as few as one variant. Each node behaves in the manner described above: it performs a discrete-event simulation of the design variants with which it is tasked, computes an optimum time-series, and uses DTW to compare the individual time-series with the optimum. Note that the optimum time-series is now a *local optimum* since the other nodes will perform the same tasks on their design variants. In addition, for each node, BRTA calculates the range (minimum-to-maximum) for the time-series computed locally, as well as the normalized distance (distance divided by the number of points in the time-series). For example, Figure 4 shows the time-series captured by *eQual* for the *Reliability* of two of Hadoop's variants. The left variant's normalized distance from its local maximum is 0.04, and the right variant's 0.35.

Instead of returning all simulation-generated data to the assessment node (recall Figure 3), BRTA only sends a summary containing the above measurements. The global assessment algorithm gathers these summaries and calculates the distance to the global optimum time-series for each time-series using the following formula:

$$D_g = \begin{cases} Max_g - O_l + D_l & (\textit{if direct}) \\ O_l - Min_g + D_l & (\textit{if inverse}) \end{cases}$$

D_g is the distance to the global optimum; O_l is the local optimum; D_l is the distance to the local optimum; Max_g (Min_g) is the global max (min) value among all time-series of a non-functional property.[1]

[1]The unblinded version of this paper will include a link to a technical report with a proof of this formula's correctness.

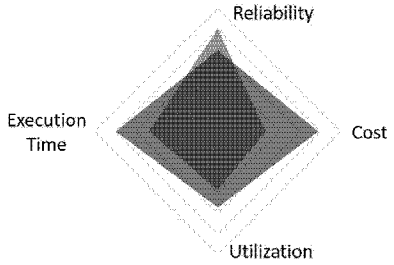

Figure 5: *eQual*'s visualization of two candidate Hadoop variants showing their respective values for the four properties of interest.

The updated summaries now include the globally normalized values for each time-series in each variant and will be used to rank the variants. To use the D_g values of different non-functional properties to calculate the overall utility of a design variant, we linearly rescale them to a value between 0 and 1. The overall quality of the system, then, is the average, weighted by the importance coefficients provided by the architects (recall Section 3.2), among all of these values. In cases when multiple variants have comparable qualities, *eQual* also allows architects to visually compare them. Figure 5 shows an example such a visualization: the overlapping radar diagrams provide architects with details that are lost in the single system-quality numerical values.

4 EVALUATION

We evaluate *eQual* to answer three questions. (1) How onerous is it for architects to answer the preparation questions *eQual* asks? (2) Can *eQual* find solutions that are close to optimal? (3) Can *eQual* scale to models with large numbers of variation points? More specifically, we aim to measure *eQual*'s *usability*, *effectiveness*, and *scalability*. To evaluate *eQual*, we have implemented its prototype on top of DomainPro [62]. The resulting extension to DomainPro totals 4.7K C# and 1.0K JavaScript SLoC. Furthermore, to enable an extensive evaluation of *eQual*'s effectiveness, we built a utility totaling an additional 1,000 C# and 200 MATLAB SLoC, as detailed in Section 4.2.

4.1 *eQual*'s Usability

The focus of the usability evaluation is to measure how onerous *eQual* is to apply by architects in practice. The assumption we make in this evaluation is that *eQual* provides high-quality results. The actual quality of those results will be evaluated empirically in Section 4.2. We first present an analytical argument for *eQual*'s usability, and then the results of its empirical evaluation.

A. Analytical Argument

In Section 3.2, we explained the questions *eQual* asks of architects. Let us assume that a system has N_v variation points and N_p properties. For each of them, *eQual* asks a three-part question. The maximum number of field entries *eQual* requires an architect to make is, therefore, $3 \times (N_v + N_p)$. Recall that the architect also has the option of not answering some (or even all) of the questions regarding variation points.

As discussed above, our analysis of Hadoop has relied on previously identified four critical non-functional properties [9, 10]. Prior research suggests that there are usually 4–6 non-functional properties of interest in a software project, and rarely more than 10 [4]. Moreover, the analysis presented in a recent study [63] showed that the number of variation points per Hadoop version ranged between 1 and 12 [63]. Taking the largest number of variation points for a single Hadoop version and the four properties, an architect using *eQual* would have to provide no more than 48 answers to explore the 4-dimensional decision space of at least 2^{12} system variants.

The number of individual answers an architect is expected to provide is precisely bounded by *eQual*. The granularity of each answer is very small and its cognitive complexity is low. Most importantly, *eQual* does not place a new burden on architects, but only makes explicit the information that architects already have to consider during system design.

B. Empirical Comparison to the State-of-the-Art

In addition to the analytical argument provided above, we also modeled Hadoop in GuideArch [24], the most closely related approach for exploring alternative design decisions. We compared the *eQual* and GuideArch models in terms of the numbers of field entries and the time required to complete them.

GuideArch helps architects make decisions in the early stages of software design using fuzzy math [77] to deal with uncertainties about system variation points in a quantifiable manner. To apply fuzzy math, GuideArch uses three-point estimates, asking architects to provide pessimistic, most likely, and optimistic values to describe the effects of their decisions on the system's non-functional properties. For instance, in the case of Hadoop's *Processing Power*, for each decision (such as using machines with 2Ghz CPUs) architects have to specify the pessimistic, optimistic, and most likely values for the *Reliability*, *Utilization*, *Execution Time*, and *Cost* properties.

GuideArch does not require the creation of a system model. However, GuideArch's usefulness is contingent on the accuracy of its inputs, which requires in-depth knowledge of the system's domain and behavior. This level of expertise is rarely available. To alleviate this issue, GuideArch's authors suggest that architects obtain this information by analyzing prior data, looking at similar systems, studying manufacturer specifications, and reading scientific publications [24]. These are difficult and time-consuming tasks that are likely to rival the design effort required by *eQual*.

The specification of non-functional properties in GuideArch is similar to *eQual*. However, as discussed above, the specification of variation points is different, which, in turn, impacts the modeling and analysis of available options. GuideArch requires that all options be specified discretely, and is unable to explore ranges.

In a representative experiment, we selected five options for each of Hadoop's variation points from Figure 2, totaling 25 alternatives. For example, instead of simply specifying the range 10–100 for *Pool Size*, we explicitly provided 10, 25, 50, 75, and 100 as the options. The next step was to specify how each decision affects the non-functional properties of the system. In doing so, we had to fill in a 25×12 matrix in GuideArch. For *eQual*, we had to answer 27 questions: 3×5 for the five variation points, and 3×4 for the four non-functional properties. Overall, it took one of this paper's authors more than four hours to complete over 300 mandatory fields in GuideArch. By contrast, it took the same author under six minutes to answer the 27 questions required by *eQual*.

This discrepancy only grows for larger problems (e.g., more variation points or more GuideArch options within a given variation point). In general, if T_f is the total number of field entries in GuideArch, N_P the number of properties, N_V the number of variation points, and a_i the number of alternatives for a variation point v_i, then

$$T_f = 3N_P \sum_{i=1}^{N_V} a_i$$

The number of GuideArch field entries grows quadratically in the number of properties and variation points. The number of field entries in *eQual* grows linearly: $3\times(N_v + N_p)$. This results in a footprint for *eQual* that is orders of magnitude smaller than GuideArch's when applied to large systems.

4.2 *eQual*'s Effectiveness

Most other approaches concerned with design quality (e.g., DomainPro [62], Palladio [5], ArchStudio [17], XTEAM [22], PCM [46]), focus on a single variant, and do not explore the space of possible design decisions (see Section 5 for further discussion). They are complementary to *eQual*, as *eQual* can use each of them to evaluate the individual variants' quality; our prototype *eQual* implementation uses DomainPro for this purpose. Prior work has shown that those techniques that aid engineers with arriving at effective designs for their systems (e.g., ArchDesigner [1]) underperform GuideArch in the quality of their top-ranked designs [23].

For the these reasons, we evaluated *eQual*'s effectiveness by directly comparing it with GuideArch as the leading, state-of-the-art approach. Unlike prior work in the area, which has traditionally been limited to such head-to-head comparisons [1, 24], we also assessed the quality of *eQual*'s results on systems with known optimal configurations. Our evaluation indicates that *eQual* produces effective designs, which are of higher quality than prior work.

A. Head-to-Head Comparison with the State-of-the-Art

Both eQual and GuideArch use known optimization methods. Their absolute effectiveness is difficult to determine as it requires ground-truth results for the modeled systems, but we can compare their effectiveness relative to one another.

To that end, we analyzed the Hadoop models created with *eQual* and GuideArch as described in Section 4.1 and compared the top-ranked variants they returned. For example, in the case of the experiment highlighted in Section 4.1, both tools produced Hadoop variants that were equally reliable (94%), had equal machine utilization (99%), and comparable cost (17 for GuideArch vs. 19 for *eQual*). However, *eQual*'s top-ranked variant was nearly 7.5 times faster than GuideArch's (154s vs. 1,135s.). We identified one possible reason for this discrepancy: We observed that GuideArch consistently selects variants with lower machine reliability but higher redundancy than those selected by *eQual*. The exact reasons behind this strategy are unclear from GuideArch's publications; we have contacted its authors for possible answers.

We acknowledge that, as creators of *eQual*, we are more familiar with it than with GuideArch. However, the author who performed the analysis has extensive experience with architectural modeling, including with GuideArch. Furthermore, we have a good understanding of Hadoop and made every effort to use GuideArch fairly. Our observations are buttressed by the fact that the quality of the variants GuideArch recommends depends heavily on the architect's ability to predict the effects of the design decisions on the system's non-functional properties. This is a non-trivial, error-prone task regardless of one's familiarity with GuideArch.

B. Evaluation on Systems with Known Optimal Designs

We further evaluated *eQual*'s effectiveness against known fitness models of six real-world systems, summarized in Figure 6. Fitness models describe the non-functional properties of a system using its variation points and their interactions. The fitness models we used in our evaluation were obtained by Siegmund et al. [64, 65] and shown to accurately predict the non-functional properties of highly configurable systems. These fitness models aim to detect interactions among options (or features) and evaluate their influence on the system's non-functional attributes. Each has been obtained by numerous measurements of different variants of a software system. We decided to use these fitness models because they are analogous to our objective in *eQual*, despite being applied to software systems at a later stage (i.e., systems that are already implemented and deployed). Furthermore, the resulting subject systems' decision spaces range from 100,000 variants to 1 quadrillion variants, making them attractive for testing *eQual*'s range of applicability.

System	Domain	Var. Points	Terms	Size
Apache	Web Server	11	9	10^6
BDB C	Berkeley DB C	9	10	10^5
BDB J	Berkeley DB Java	6	8	10^6
Clasp	Answer Set Solver	10	17	10^{15}
LLVM	Compiler Platform	7	8	10^7
AJStats	Analysis Tool	3	4	10^7

Figure 6: Software systems used to study the effectiveness of *eQual*. *Var. Points* is the number of variation points in each system; *Terms* is the number of terms in the systems' fitness models; and *Size* is the total number of variants in the design space.

		Random		Edge-Case	
System	Default	Mean	σ	Mean	σ
Apache	0.264	0.311	0.146	0.899	0.163
BDB C	0.763	0.564	0.325	0.983	0.035
BDB J	0.182	0.517	0.408	1.000	0.000
Clasp	0.323	0.352	0.174	0.859	0.179
LLVM	0.253	0.235	0.234	0.902	0.219
AJStats	0.856	0.780	0.269	0.963	0.048
Overall	0.440	0.460	0.592	0.934	0.107

Figure 7: Comparison between the two seeding strategies employed by *eQual* and the quality of the nominal solutions commonly selected by architects. *Default* depicts the common solutions used by architects, obtained from [65].

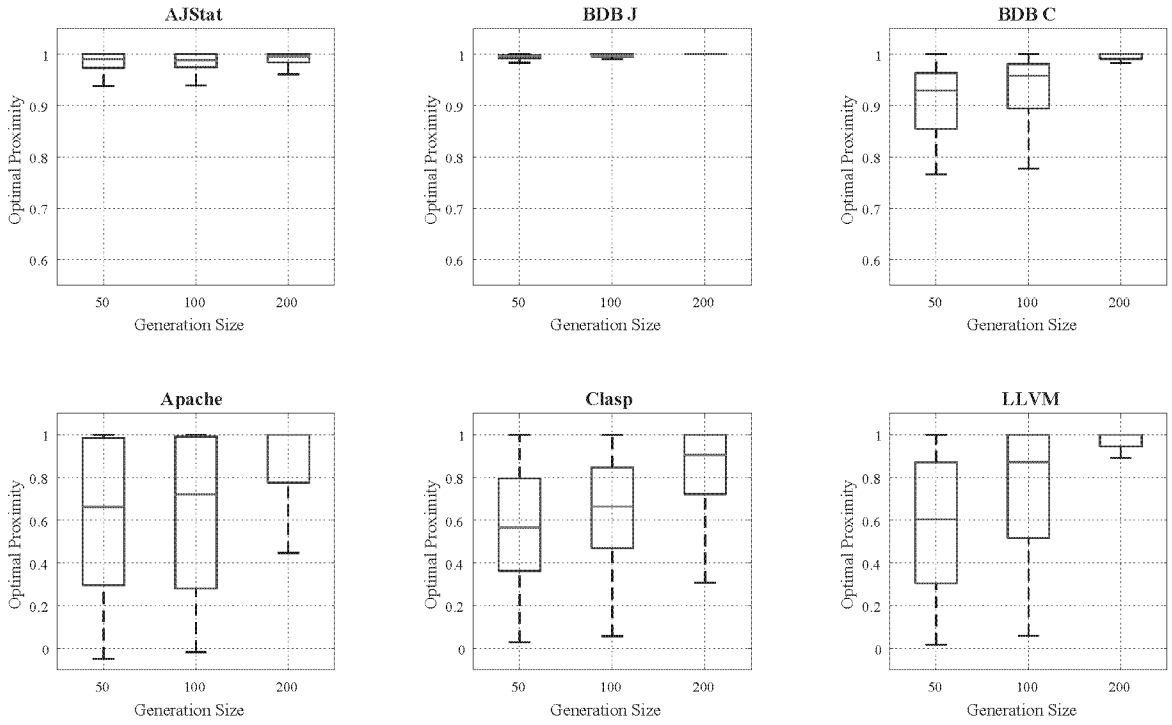

Figure 8: *Optimal Proximity* distribution of the best variant generated by *eQual* using generation sizes of 50, 100, and 200. Each box plot comprises 100 executions of *eQual*'s exploration function. The y-axes of top three systems range 0.6–1, whereas the y-axes on the bottom three systems legend start at 0. This is done to enable better visualization of the variants' quality distributions for the top three systems.

Conceptually, a fitness model is simply a function from variants to a fitness measurement $\Omega : C \rightarrow \mathbb{R}$, where fitness can be an aggregation of any measurable non-functional property that produces interval-scaled data. The model is described as a sum of terms over variation option values. Individual terms of the fitness model can have different shapes, including $n.c(X)$, $n.c(X)^2$, or $n.c(X).\sqrt{c(Y)}$. For illustration, a configurable database management system with the options encryption (E), compression (C), page size (P), and database size (D) may have the following fitness model:

$$\Omega(c) = 50 + 20.c(E) + 15.c(C) - 0.5c(P) + 2.5.c(E).c(C).c(D)$$

In general, the fitness models are of the following form:

$$\Omega(c) = \beta_0 + \sum_{i..j \in O} \Phi(c(i)..c(j))$$

β_0 represents a minimum constant base fitness shared by all variants. Each term of the form $\Phi(c(i)..c(j))$ captures an aspect of the overall fitness of the system.

Because only aggregate fitness models were available to us, without loss of generality, we treated each term as an individual non-functional property of a given system, and translated its coefficients into *eQual*'s coefficients. Then, using the formula of each term, we generated the corresponding constant time-series representing the term. These time-series were subsequently passed to *eQual* for exploration. To measure *eQual*'s effectiveness, we normalized each variant's fitness and calculated the fitness of the best variant found by *eQual* using the ground-truth fitness models. We then calculated that variant's distance from the global optimum. We call this the *Optimal Proximity*. These steps were accomplished via an extension to *eQual* totaling 1,000 C# SLoC, and an additional 200 MATLAB SLoC to tune and visualize the resulting *eQual* models.

Figures 7 and 8 depict the results of applying *eQual* on our six subject systems using the two strategies discussed in Section 3.3: random seeding and edge-case seeding. Figure 7 compares *eQual*'s two strategies against the solutions yielded by using the default values suggested by the six systems' developers [65]. These results were obtained by setting the cross-over ratio for the genetic algorithm to 0.85 and the mutation rate to 0.35, using 4 generations of size 200. These hyper-parameters were obtained over nearly 30,000 test executions, by using grid-search to find the most suitable parameters on average. The results in Figure 7 show that in most cases even the purely random seeding strategy for *eQual* is at least as effective as the default values suggested by the developers. On the other hand, the edge-case strategy finds superior variants that on average exhibit over 93% of the global optimum. Figure 8 provides additional detail, showing the distribution of running *eQual* on the six subject systems 100 times using the edge-case strategy, with generation sizes of 50, 100, and 200. The figure shows that, with a larger number of generations, *eQual* is able to produce variants that, on average, tend to match the reported global optimum for each system; in the case of Clasp, the lone exception, the quality of *eQual*'s suggested variant was still over 90% of the global optimum.

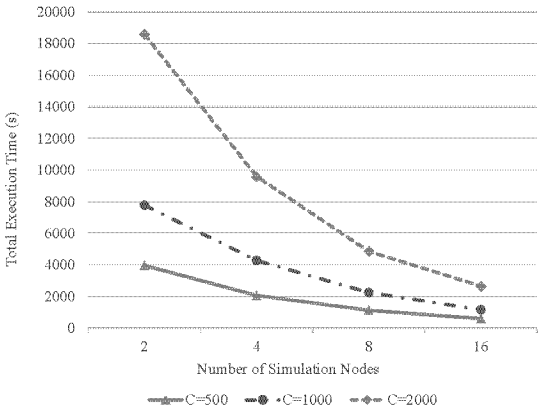

Figure 9: *eQual*'s scalability with respect to the number of simulation nodes.

4.3 *eQual*'s Scalability

To evaluate *eQual*'s scalability, we used the Google Compute Engine (GCE). We created 16 n1-standard-1 nodes (the most basic configuration available in GCE, with 1 vCPU and 3.75 GB memory) as simulation nodes, and a single n1-standard-2 node (2 vCPU and 7.5 GB memory) as the central controller node. All nodes were located in Google's us-central1-f datacenter. We used the variation points and non-functional property descriptions described in Section 3.

Number of Nodes — The first aspect of *eQual*'s scalability we evaluated is the observed speed-up when increasing the number of employed simulation nodes. We used the general genetic algorithm for 8 generations and the generation size of 256 variants, totaling 2,048 variants. We did this with 2, 4, 8, and 16 simulation nodes, and for three values of the *Computation Size* variation point. The execution time was inversely proportional to the number of nodes (Figure 9), suggesting that our approach can be scaled up optimally by adding more nodes. Using more powerful nodes can further speed up the computation. Note that each data point in Figure 9 consists of 2,048 simulations. Overall, we simulated more than 24,500 design variants.

Number of Events — For the second part of the scalability evaluation, we measured the impact of increasing the number of events that are generated during a simulation. This is indicative of how

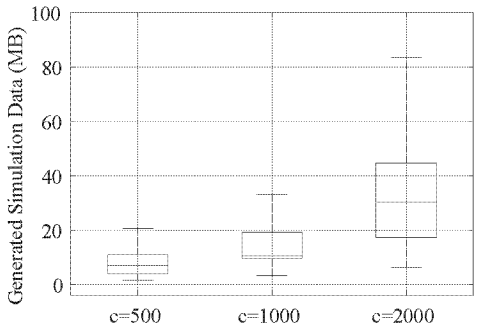

Figure 10: Sizes of data files generated by simulations for different values of *Computation Size* (*c*).

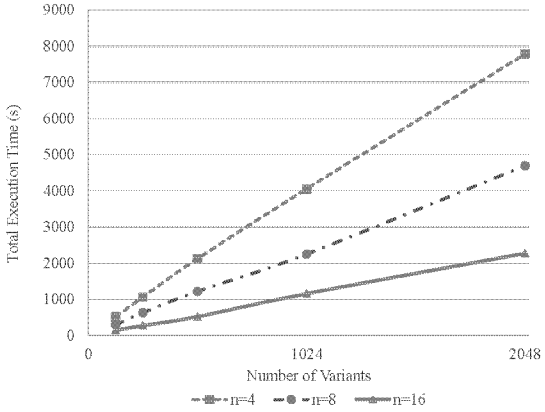

Figure 11: *eQual*'s scalability with respect to the number of variants.

well *eQual* performs on larger models. The total number of events generated in Hadoop is nondeterministic. However, based on the characteristics of the model, we hypothesized that increasing the *Computation Size* should increase the number of events roughly linearly if other variation points remain unchanged. We empirically evaluated this hypothesis by using the average sizes of the time-series object files generated during simulation as a reasonable proxy for the number of events. Figure 10 shows that, on average, the total number of events is directly proportional to *Computation Size*. Coupled with the performance that *eQual* demonstrated for the same values of *Computation Size*, shown in Figure 9 and discussed above, this is indicative of *eQual*'s scalability in the face of growing numbers of simulation events.

Number of Variants — Finally, we studied *eQual*'s performance in the face of growing numbers of design variants. We modified the genetic algorithm configurations to use five generation sizes: 16, 32, 64, 128, and 256. For each generation size, we ran *eQual* for 8 generations, on 4, 8, and 16 nodes. Figure 11 shows that *eQual* is able to analyze over 2,000 design variants in ~120 minutes on 4 nodes, with a speed-up linear in the number of nodes, down to ~30 minutes on 16 nodes.

4.4 Threats to Validity

While the evaluation of *eQual* indicates that it can easily scale, has a small footprint, and finds accurate solutions, we also took several steps to mitigate the possible threats to our work's validity. Two constituent parts of *eQual* help mitigate the threats to its **construct validity**: (1) the dynamic analysis of system designs by simulation and (2) creating assessment models based on DTW. In the past, these two solutions have been extensively used to great success. Discrete event simulations have been used in a plethora of domains (e.g., avionics, robotics, healthcare, computer networks, finance, etc.) [67, 69], and the bedrock of our assessment models, DTW, is so prevalent that it is "difficult to overstate its ubiquity" [19, 57]. DTW also subsumes Euclidean distance [19] as a special case [57], which increases the *eQual*'s range of applicability. The threat to the **external validity** of our work is mitigated by the use of an

MDA-based approach. MDA solutions have been shown to be sufficiently robust and scalable, and are widely used in research and industry [32, 37].

5 RELATED WORK

DomainPro [61, 62], Palladio [5], ArchStudio [17], XTEAM [22], PCM [46], and other tools allow simulation-based analysis of a single variant's architectural model. By contrast, *eQual* explores many variants to help make design decisions.

Rule-based approaches identify problems in a software design model and apply rules to repair them. MOSES uses stepwise refinement and simulation for performance analysis [15]. ArchE helps meet the quality requirements during the design phase by supporting modifiability and performance analysis, and suggesting potential design improvements [47]. DeepCompass explores the design space of embedded systems, relying on the ROBOCOP component model and a Pareto analysis to resolve the conflicting goals of optimal performance and cost between different architecture candidates [6]. Parsons et al. [54] introduced an approach for detecting performance anti-patterns in Java-EE architectures. Their method requires an implementation of a component-based system, which can be monitored for performance. Since the approach is meant to improve existing systems, it cannot be used during early development stages. Their approach is meant to improve existing systems and cannot be used during early development stages. PUMA facilitates communication between systems designed in UML and non-functional property prediction tools [75], focusing on performance and support feedback loop specification in JESS [76]. FORMULA aims to overcome the challenge of the size of the design space in DSE by exploring only a small subset of the space by removing the design candidates considered equivalent based on a user-defined notion of equivalence [34]. This heavily relies on "correct" estimates of non-functional attributes of the model from the users. Unlike *eQual*, these approaches are also limited by their predefined rules and cannot explore the complete design space.

Metaheuristic approaches treat architecture improvement as an optimization problem. DeSi [44], ArcheOpterix [2], and PerOpteryx [39] use evolutionary algorithms to optimize a system's deployment with respect to quality criteria. Multi-criteria genetic algorithms can automatically improve software architecture based on trade-off analyses, but existing approaches (e.g., [46]) tend to suffer from scalability issues. AQOSA provides modeling based on AADL and performance analysis tools, and evaluates design alternatives based on cost, reliability, and performance [41]. The SASSY framework targets service-oriented architecture models and selects services and a pattern application to fulfill the quality requirements [50]. Metaheuristic simulation based on a genetic algorithm can derive the deployment architecture and runtime reconfiguration rules to move a legacy application to the cloud environment [25]. Mixed-integer linear programming can find minimum cost configuration for a given cloud-based application [3]. DESERT explores design alternatives by modeling system variations in a tree structure and using Boolean constraints to cut branches without feasible solutions [21]. DESERT-FD automates the constraint generation process and design space exploration [21]. GDSE uses meta-programming of domain-specific design space exploration problems and expresses constraints for solvers to generate architectural solutions [58]. GuideArch is the most closely related technique for exploring design decisions [24], but it relies on a much more burdensome fuzzy math system specifications [77] than *eQual*'s requirements (recall Section 4.1). *eQual* similarly uses metaheuristic search to find architectural solutions within a large design space, but focuses on non-functional properties and and requires less work by the architects to use.

Software Product Lines (SPLs) typically target a fundamentally different problem than *eQual*. SPLs [14, 30, 60, 66] allow for product derivation, the process of configuring reusable software artifacts for a set of requirements. Unlike SPLs, *eQual* neither adds nor removes features from a product. SPLs can use genetic algorithms to optimize feature selection [29], but, unlike *eQual*, this requires developers to create objective functions to measure each variant's fitness. Combining multi-objective search and constraint solving allows configuring large SPLs [31], but unlike *eQual*, this requires using historical data about the SPL's products to evaluate variants, which prevents it from being used for new systems. Optimizing highly configurable systems, despite being aimed at fully implemented and deployed software systems, has clear relations to *eQual*. Among these techniques we used the studies conducted by Siegmund et al. [64, 65] to evaluate the effectiveness of *eQual*. Oh et al. [53], and Sayyad [59] have recently devised techniques to more efficiently explore the space of the system configurations. These techniques can complement *eQual*'s exploration strategies.

6 CONTRIBUTIONS

It is widely acknowledged that, in a software engineering project, early architectural design decisions are usually the most impactful. However, this perceived criticality of the early decisions is not reflected in the support available to architects for making and evaluating them. The work described in this paper provides an important step in addressing the chasm between the needed and available support in this area. Our approach, *eQual*, guides architects in making informed choices, by quantifying the consequences of their decisions early and throughout the design process. *eQual* does so in a way that aims to minimize additional burden on the architects, instead only providing structure and automated support to the architects' already existing tasks. *eQual* is able to solve very large problems efficiently, guiding architects to select high-quality architectural variants for their system.

While our results show promise, research challenges remain to improving *eQual*'s practical effectiveness. Our work to date has assumed that architects know the relative importance of the non-functional properties in their systems. *eQual* allows architects to change the non-functional properties' importance coefficients, but future versions will actively guide architects in the identification of design hot-spots and help their understanding of the relative importance of the properties. Combining *eQual* with a software architecture recovery technique will extend its applicability to existing software systems with legacy architectures, for their future development and improvements. Our successful application of *eQual* to the six implemented real-world systems with known fitness models [64, 65] provides further confidence in the likely success of this strategy.

REFERENCES

[1] Tariq Al-Naeem, Ian Gorton, Muhammed Ali Babar, Fethi Rabhi, and Boualem Benatallah. A quality-driven systematic approach for architecting distributed software applications. In *International Conference on Software Engineering (ICSE)*, pages 244–253, 2005.

[2] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering (TSE)*, 39(5):658–683, 2013.

[3] Danilo Ardagna, Giovanni Paolo Gibilisco, Michele Ciavotta, and Alexander Lavrentev. A multi-model optimization framework for the model driven design of cloud applications. In *Search-Based Software Engineering*, pages 61–76. Springer, 2014.

[4] Jagdish Bansiya and Carl G Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering (TSE)*, 28(1):4–17, 2002.

[5] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[6] Egor Bondarev, Michel RV Chaudron, and Erwin A de Kock. Exploring performance trade-offs of a jpeg decoder using the deepcompass framework. In *Proceedings of the 6th international workshop on Software and performance*, pages 153–163. ACM, 2007.

[7] Bas Boone, Sofie Van Hoecke, Gregory Van Seghbroeck, Niels Joncheere, Viviane Jonckers, Filip De Turck, Chris Develder, and Bart Dhoedt. Salsa: Qos-aware load balancing for autonomous service brokering. *Journal of Systems and Software*, 83(3):446–456, 2010.

[8] Aleksandr Alekseevich Borovkov. *Asymptotic methods in queuing theory*. John Wiley & Sons, 1984.

[9] Yuriy Brun, George Edwards, Jae young Bang, and Nenad Medvidovic. Smart redundancy for distributed computation. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 665–676, Minneapolis, MN, USA, June 2011.

[10] Yuriy Brun, Jae young Bang, George Edwards, and Nenad Medvidovic. Self-adapting reliability in distributed software systems. *IEEE Transactions on Software Engineering (TSE)*, 41(8):764–780, August 2015.

[11] Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics & Optimization*, 4(1):41–59, 1977.

[12] Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *International Conference on Software Engineering (ICSE)*, pages 362–371. ACM, 2005.

[13] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, 2007.

[14] Thelma Elita Colanzi, Silvia Regina Vergilio, Itana Gimenes, and Willian Nalepa Oizumi. A search-based approach for software product line design. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 237–241. ACM, 2014.

[15] Vittorio Cortellessa, Pierluigi Pierini, Romina Spalazzese, and Alessio Vianale. Moses: Modeling software and platform architecture in uml 2 for simulation-based performance analysis. In *Quality of Software Architectures. Models and Architectures*, pages 86–102. Springer, 2008.

[16] Marco D'Ambros, Alberto Bacchelli, and Michele Lanza. On the impact of design flaws on software defects. In *QSIC 2010 (10th International Conference on Quality Software)*, pages 23–31. IEEE, 2010.

[17] Eric Dashofy, Hazel Asuncion, Scott Hendrickson, Girish Suryanarayana, John Georgas, and Richard Taylor. Archstudio 4: An architecture-based meta-modeling environment. In *International Conference on Software Engineering (ICSE) Demo track*, pages 67–68, 2007.

[18] Pablo de Oliveira Castro, Stéphane Louise, and Denis Barthou. Reducing memory requirements of stream programs by graph transformations. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 171–180. IEEE, 2010.

[19] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[20] Christoph Dorn, George Edwards, and Nenad Medvidovic. Analyzing design tradeoffs in large-scale socio-technical systems through simulation of dynamic collaboration patterns. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 362–379. Springer Berlin Heidelberg, 2012.

[21] Brandon K Eames, Sandeep K Neema, and Rohit Saraswat. Desertfd: a finite-domain constraint based tool for design space exploration. *Design Automation for Embedded Systems*, 14(1):43–74, 2010.

[22] George Edwards, Yuriy Brun, and Nenad Medvidovic. Automated analysis and code generation for domain-specific models. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 161–170. IEEE, 2012.

[23] Naeem Esfahani. *Management of uncertainty in self-adaptive software*. PhD thesis, George Mason University, 2014.

[24] Naeem Esfahani, Sam Malek, and Kaveh Razavi. GuideArch: Guiding the exploration of architectural solution space under uncertainty. In *International Conference on Software Engineering (ICSE)*, pages 43–52, 2013.

[25] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *International Conference on Software Engineering (ICSE)*, pages 512–521. IEEE Press, 2013.

[26] Walter R Gilks. *Markov chain monte carlo*. Wiley Online Library, 2005.

[27] Swapna S Gokhale. Software application design based on architecture, reliability and cost. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 2, pages 1098–1103. IEEE, 2004.

[28] Lars Grunske, Peter Lindsay, Egor Bondarev, Yiannis Papadopoulos, and David Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In *Architecting dependable systems IV*, pages 188–209. Springer, 2007.

[29] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208–2221, 2011.

[30] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41(4), 2008.

[31] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *International Conference on Software Engineering (ICSE)*, volume 1, pages 517–528. IEEE, 2015.

[32] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *International Conference on Software Engineering (ICSE)*, pages 471–480. IEEE, 2011.

[33] IBM. Ibm rationale rhapsody. http://www-03.ibm.com/software/products/en/ratirhapfami.

[34] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An approach for effective design space exploration. In Radu Calinescu and Ethan Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 33–54, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[35] Rick Kazman, Jai Asundi, and Mark Klein. Quantifying the costs and benefits of architectural decisions. In *International Conference on Software Engineering (ICSE)*, pages 297–306, 2001.

[36] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.

[37] Anneke G Kleppe, Jos Warmer, Wim Bast, and MDA Explained. *The model driven architecture: practice and promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[38] Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer Berlin Heidelberg, 1989.

[39] Anne Koziolek. *Automated improvement of software architecture models for performance and other quality attributes*, volume 7. KIT Scientific Publishing, 2014.

[40] Daniel R. Levinson. An overview of 60 contracts that contributed to the development and operation of the federal marketplace, oei-03-14-00231. http://oig.hhs.gov/oei/reports/oei-03-14-00231.pdf, August 2014.

[41] Rui Li, Ramin Etemaadi, Michael TM Emmerich, and Michel RV Chaudron. An evolutionary multiobjective optimization approach to component-based software architecture design. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 432–439. IEEE, 2011.

[42] FMS Luke Chung. Healthcare.gov is a technological disaster. http://goo.gl/8B1fcN, 2013.

[43] Jiefei Ma, Franck Le, Alessandra Russo, and Jorge Lobo. Declarative framework for specification, simulation and analysis of distributed applications. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1489–1502, 2016.

[44] Sam Malek, Nenad Medvidovic, and Marija Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering (TSE)*, 38(1):73–100, 2012.

[45] Marzio Marseguerra, Enrico Zio, and Luca Podofillini. Genetic algorithms and monte carlo simulation for the optimization of system design and operation. In *Computational Intelligence in Reliability Engineering*, pages 101–150. Springer, 2007.

[46] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *International Conference on Performance Engineering (WOSP/SIPEW)*, pages 105–116, 2010.

[47] John D McGregor, Felix Bachmann, Len Bass, Philip Bianco, and Mark Klein. Using arche in the classroom: One experience. Technical report, DTIC Document, 2007.

[48] Gianantonio Me, Coral Calero, and Patricia Lago. Architectural patterns and quality attributes interaction. In *IEEE Workshop on Qualitative Reasoning about Software Architectures (QRASA)*. IEEE, 2016.

[49] Nenad Medvidovic and Richard N Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering (TSE)*, 26(1):70–93, 2000.

[50] Daniel A Menascé, John M Ewing, Hassan Gomaa, Sam Malex, and João P Sousa. A framework for utility-based service oriented design in sassy. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 27–36. ACM, 2010.

[51] Joaquin Miller, Jishnu Mukerji, M Belaunde, et al. Mda guide. *Object Management Group*, 2003.

[52] Tim Mullaney. Demand overwhelmed healthcare.gov. http://goo.gl/k3o4Rg, 2013.

[53] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 61–71, New York, NY, USA, 2017. ACM.

[54] Trevor Parsons. *Automatic detection of performance design and deployment antipatterns in component based enterprise systems*. PhD thesis, Citeseer, 2007.

[55] Carlo Poloni and Valentino Pediroda. Ga coupled with computationally expensive simulations: tools to improve efficiency. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 267–288, 1997.

[56] Pasqualina Potena. Composition and tradeoff of non-functional attributes in software systems: research directions. In *Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 583–586. ACM, 2007.

[57] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 7(3):10, 2013.

[58] Tripti Saxena and Gabor Karsai. Mde-based approach for generalizing design space exploration. In *Model Driven Engineering Languages and Systems*, pages 46–60. Springer, 2010.

[59] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 465–474, Nov 2013.

[60] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *International Conference on Software Engineering (ICSE)*, pages 492–501. IEEE Press, 2013.

[61] Ognjen Scekic, Christoph Dorn, and Schahram Dustdar. Simulation-based modeling and evaluation of incentive schemes in crowdsourcing environments. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 167–184. Springer Berlin Heidelberg, 2013.

[62] Arman Shahbazian, George Edwards, and Nenad Medvidovic. An end-to-end domain specific modeling and analysis platform. In *International Conference on Software Engineering (ICSE)*. IEEE, 2016.

[63] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. Recovering architectural design decisions. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2018.

[64] N. Siegmund, S. S. Kolesnikov, C. KÃďstner, S. Apel, D. Batory, M. RosenmÃijller, and G. Saake. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 167–177, June 2012.

[65] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 284–294, New York, NY, USA, 2015. ACM.

[66] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3-4):487–517, 2012.

[67] Ghanem Soltana, Nicolas Sannier, Mehrdad Sabetzadeh, and Lionel C Briand. A model-based framework for probabilistic simulation of legal policies. In *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*, pages 70–79. IEEE, 2015.

[68] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[69] Atul Thakur, Ashis Gopal Banerjee, and Satyandra K Gupta. A survey of cad model simplification techniques for physics-based simulation applications. *Computer-Aided Design*, 41(2):65–80, 2009.

[70] Nikola Trčka, Martijn Hendriks, Twan Basten, Marc Geilen, and Lou Somers. Integrated model-driven design-space exploration for embedded systems. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 339–346. IEEE, 2011.

[71] United States Government Accountability Office. Report to congressional requester, gao-15-238. http://www.gao.gov/assets/670/668834.pdf, 2015.

[72] US Centers for Medicare and Medicaid Services. Mckinsey and co. presentation on health care law. http://goo.gl/Nns9mr, 2013.

[73] US Department of Health and Human Services. Healthcare.gov progress and performance report. http://goo.gl/XJRC7Q, 2013.

[74] Andreea Vescan. A metrics-based evolutionary approach for the component selection problem. In *Computer Modelling and Simulation, 2009. UKSIM'09. 11th International Conference on*, pages 83–88. IEEE, 2009.

[75] Murray Woodside, Dorina C Petriu, Dorin B Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer. Performance by unified model analysis (puma). In *Proceedings of the 5th international workshop on Software and performance*, pages 1–12. ACM, 2005.

[76] Jing Xu. Rule-based automatic software performance diagnosis and improvement. *Performance Evaluation*, 69(11):525–550, 2012.

[77] Hans-Jürgen Zimmermann. *Fuzzy set theory and its applications*. Springer Science & Business Media, 2011.