eQual: Informing Early Design Decisions

Anonymous Author(s)

ABSTRACT

When designing a software system, architects make a series of design decisions that directly impact the system's quality. The number of available design alternatives grows rapidly with system size, creating an enormous space of intertwined design concerns that renders manual exploration impractical. We present *eQual*, a model-driven technique for simulation-based assessment of architectural designs. While it is impossible to guarantee optimal decisions so early in the design process, *eQual* improves decision quality. *eQual* is effective in practice because it (1) limits the amount of information the architects have to provide and (2) adapts optimization algorithms to effectively explore massive spaces of design alternatives. A user study shows that, compared to the prior state of the art, engineers using *eQual* produce statistically significantly higher-quality designs with a large effect size, are statistically significantly more confident in their designs, and find *eQual* easier to use.

1 INTRODUCTION

For most software systems, architectural decisions are introduced incrementally, as system information becomes available. Some decisions are made quite early, but have long-lasting impact. Making suboptimal decisions can lead to system inefficiency or to a costly redesign later in the development process. This paper introduces *eQual*, a technique that helps make better-informed design decisions throughout the process by using simulation to help architects understand the implications of specific design choices using the partial information available at the time the decisions are made.

Consistently making *optimal* decisions early in the process may be impossible because only partial information about the system is known at that time. *eQual*'s goal is to compute relevant information that helps make *better* decisions than is possible using the state-of-the-art techniques. In fact, in Section 4, in a controlled user study with 15 engineers, we demonstrate engineers using *eQual* produce statistically significantly ($p < 0.001$) higher-quality designs than those who use a state-of-the-art technique [29] and those who do not use specialized tools. The effect size of the treatment is large, meaning the designs they produce using *eQual* are of much higher quality. We further show that *eQual* users are statistically significantly ($p < 0.05$) more confident in their designs and prefer *eQual*'s usability. Applied to a benchmark of systems with previously published ground-truth key design choices [79, 80], we show that *eQual* recommends variants that are either optimal or are of essentially the same quality as the optimal solutions. Finally, we show that *eQual* is *easier to use* than prior tools. Overall, we show, either directly and by transitivity via prior studies, that *eQual* outperforms existing techniques within its problem scope, including GuideArch [29], ArchDesigner [1], ArcheOpterix [2], ATAM [17], CBAM [45], Doyle [25], and Noppen et al. [66].

The problem *eQual* tackles is hard because in any real-world system, there are countless design decisions to be made [78, 83, 87, 88], early architectural designs exhibit significant uncertainties [13, 29, 50], and the decisions intertwine many factors and trade-offs that must be considered [15, 16, 69]. Ideally, architects carefully assess the individual choices to make viable design decisions that satisfy a system's requirements. However, this is frequently not done in practice [20]. A well publicized example is the Healthcare.gov (a.k.a. "Obamacare") portal, which was marred with serious problems [54, 65, 86] due to flawed architectural decisions [87, 88]: its development costs, originally estimated at ~\$100M, surpassed \$1.5B [51]. The root cause of such failures is known: evaluating design options is exceedingly complex. The space of system variants rapidly eclipses human capabilities [36]. A solution that appears to make sense intuitively may turn out to be wildly off the mark or, in the best case, suboptimal.

Architectural design decisions span diverse concerns: system structure, behavior, interaction, deployment, evolution, and non-functional properties (NFPs) [83]. Every decision is a selection of one of several possible *alternatives* for a given *variation point* in a system. For example, in a system that may have multiple authentication servers, e.g., for scalability, the servers comprise the variation point, and the specific number is an alternative. During the design of a system, an architect must make many such decisions, for example, how many replicated data stores to use, how to implement each data store (e.g., different relational or NoSQL database implementations and their relevant parameters), what implementation framework to use (e.g., different MVC-based Web frameworks and their parameters), what architectural styles to use (e.g., layered client-server that may yield an n-tiered architecture vs. event-based that may result in distributed peers), what data caching strategies to use, etc. An architecture *variant* is the set of design decisions that result in the selection of an alternative for each variation point in a system; in other words, a variant results in a *complete architecture* for the system.

eQual is a model-driven technique for simulation-based exploration and assessment of architectural designs on the cloud. *eQual*'s goal is to help compute the implications of design decisions, even when only partial information is available early in the process, suggesting an ordering on possible variants with respect to specific quality concerns. The idea is that while known information may be insufficient to make optimal decisions, some alternatives can be shown to be better than others, and that information can help engineers to explore huge spaces of design decisions to improve their choices. *eQual* asks a bounded number of relatively straightforward questions about the system under design and the engineers' preferences for the system and then automatically builds the requisite system models, distributes and runs simulations in the background, and delivers the ranked list of variants that respect the engineers' already-made design choices. Engineers can accept *eQual*'s recommendations or adjust their preferences and explore other variants.

eQual takes two inputs: (1) a software system model and (2) architects' answers to a set of questions about the parameters captured in the model. The system model is an artifact that typically already exists as part of the normal design process [83]. The questions are designed to be relatively simple to answer, and optional. Examples of questions are "What is the appropriate interaction mechanism for the selected components?" and "What is the maximum number of authentication servers in the system?" *eQual* provides interactive facilities for creating the system model in a domain-specific language (DSL). *eQual*'s questions (1) bound the search space of variants *eQual* will explore, and (2) identify the NFPs of interest. *eQual* does not assume the architects will be able to answer the questions (e.g., because of insufficient knowledge about the system) or that system parameters can be rank-ordered (e.g., because the parameters are qualitative). If the architect chooses not to answer the questions, *eQual* may explore an unbounded search space. *eQual* uses the system model and the specified bounds to generate variants, intelligently distributes those variants on a set of cloud nodes for simulation, and collects and processes the data using our novel Bipartite Relative Time-series Assessment (BRTA) algorithm. *eQual* repeats this process until it arrives at a set of satisfactory variants based on the of-interest properties.

This paper's primary contributions are:

(1) A method for automatically generating architectural assessment models from simple inputs that architects provide.
(2) The BRTA algorithm for analyzing simulation data to solve the previously prohibitively inefficient variant-assessment problem.
(3) An architecture for seamlessly distributing and parallelizing simulations to multiple computational (e.g., cloud) nodes.
(4) An ability to quickly explore and visualize many different design alternatives until a satisfactory solution is reached.
(5) An evaluation of *eQual*'s scope, ease of use, effectiveness, and scalability on real-world systems.
(6) An open-source *eQual* implementation, all evaluation scripts, and data, available at https://tinyurl.com/dse-eQual/.

In the rest of our paper, Section 2 introduces Hadoop, used to help describe and evaluate *eQual*. Section 3 describes *eQual* and Section 4 evaluates it. Section 5 places our research in the context of related work. Finally, Section 6 summarizes our contributions.

2 STATE-OF-THE-ART AND APPROACH DEMO

We first explain our choice of tools we compare our approach to, and then demonstrate our approach on a simplified system.

2.1 Choice of State-of-the-Art Comparison

Software system design is a rich research field. Architects' experience, knowledge of architectural styles and patterns, prior design of similar systems, understanding of the deployment environment, and many more attributes play a role in successful design. Existing tools can help augment the architects' knowledge to help produce better designs. This makes evaluating *eQual* challenging: it is not feasible to compare *eQual* to all different architecture tools in a single conference paper. We restrict our comparison here to tools that tackle the same problem scope as *eQual*. Our focus is neither to develop a tool that provides all the useful types of information for architects, nor to claim that the tool can produce optimal designs. Instead, *eQual*'s goal is to help make *better* decisions than using a particular set of tools designed to tackle the same problem scope.

We directly compare *eQual* to GuideArch [29] for three reasons. First, GuideArch matches *eQual*'s problem scope by aiming to compute the same type of information architects use in making design decisions. Second, prior work has demonstrated that GuideArch outperforms a large number of existing techniques in this problem scope, including ArchDesigner [1], ArcheOpterix [2], ATAM [17], CBAM [45], Doyle [25], and Noppen et al. [66]. By comparing directly with GuideArch in the same dimensions, we can be confident *eQual* outperforms these techniques as well. Third, since GuideArch's publication, despite improvement attempts [13, 30, 50, 76], GuideArch has remained the state of the art. GuideArch uses fuzzy mathematical methods to automatically select a set of near-optimal decisions from a large design space [29]. Letier et al. [13, 50] built on GuideArch to reason about uncertainty in early requirements and designs using statistical decision analysis. However, that approach only applies when the design spaces are reasonably small and amenable to exhaustive search. Our experience and a large body of literature show that real systems' designs have massive decision spaces that grow rapidly with the systems' complexity (e.g., [36, 78–80, 83]). Sedaghatbaf et al. [76] used evolutionary algorithms to explore the effects of varying the numbers of system resources and of reallocating software components across hardware hosts. These very concerns have already been considered by GuideArch (and by a GuideArch predecessor [56]), and ultimately the resulting facilities improve neither GuideArch's applicability nor its scalability. Fahmideh et al. [30] applied GuideArch's fuzzy-math approach to find an optimal set of design decisions in another domain — design-exploration of manufacturing systems — but did not improve the underlying GuideArch capabilities. As a result, GuideArch has remains the state-of-the-art approach with respect to a large number of competitors [1, 5, 13, 21, 27, 30, 50, 56, 58, 76, 77].

At the same time, GuideArch has faced two obstacles to adoption. First, it requires architects to supply potentially large amounts of information, some of which they may not readily have. Second, its effectiveness and scalability claims have only been evaluated one case study used to illustrate its features [29]. To address these limitations, we have developed and extensively evaluated *eQual*, a *model-driven*, *simulation-based* technique that tackles the same *problem space* of early architectural design for large systems, but aims to improve GuideArch's *ease of use*, *effectiveness*, and *scalability*.

What our paper demonstrates is that *eQual* can improve the process of making design decisions as compared to the tools within its problem scope. It is likely that other tools that provide other types of information useful for making design decisions can be complementary to *eQual*, as Section 5 describes. This paper provides significant evidence that *eQual* can be a part of the solution to making better, informed design decisions, but future work will certainly demonstrate ways to combine different kinds of information with *eQual*'s to further improve the process.

2.2 Using *eQual* on an Example System

We use a simplified model of Hadoop from prior work [11, 12] as a running example throughout the paper. In this model (shown in Figure 1), a *computation* is the problem being solved, consisting of many computational *tasks*. Tasks can be replicated, e.g., for reliability [12]. A *job* is an instance of a task (i.e., one replica of a task) and *machines* execute these jobs. A *task scheduler* breaks

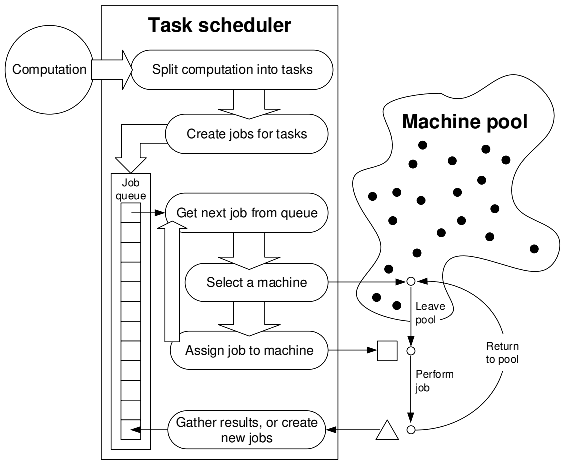

Figure 1: Hadoop system model in a domain-specific language [11].

up a computation into tasks, creates jobs as replicas of tasks, and assigns jobs to machines in the machine pool. After returning a response to the task scheduler, a machine rejoins the machine pool and can be selected for a new task.

Although *eQual* has been used to analyze Hadoop's entire design space (over 100 design decisions [78]), for simplicity of exposition, we highlight five *variation points* that affect Hadoop's key NFPs: (1) *Computation Size*, the number of tasks required to complete a computation; (2) *Redundancy Level*, the number of machines that run identical jobs; (3) *Pool Size*, the number of available machines; (4) *Machine Reliability*, the probability of a machine returning the correct result; and (5) *Processing Power* of each machine. We will discuss other, non-numerical variation points below. Figure 2 depicts representative bounds for the five variation points obtained from the previously published analysis [11, 12].

To use *eQual*, an architect specifies a design model (these models typically already exist as part of normal design activities [71, 83]) and answers a set of simple design questions, such as "What are the the lower and upper bounds on the number of system's *Pool Size*?" *eQual* provides interactive facilities for specifying this model and answers. Next, *eQual* automatically generates a set of system variants and uses discrete-event simulation to evaluate their impact on quality measures (specified by the architect as part of the model). The

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 & representative bounds [11, 12].

results inform *eQual*'s selection of a next set of variants to explore. *eQual* iterates this way until it arriving at a list of variants that optimize desirable quality measures, and presents the architect with the list to help understand the impact of the explored design decisions.

3 THE *eQual* APPROACH

eQual explores a system's design space via four steps: (1) modeling, (2) preparation, (3) selection, and (4) assessment. Steps (1) and (2) are interactive and help the architect generate *eQual*'s inputs: a design model and answers to a set of design-related questions. Steps (3) and (4) are automated and produce a list of ranked system variants. Steps (1) and (3) adapt existing solutions, while (2) and (4) are new contributions introduced by *eQual*. Steps (3) and (4) take place iteratively: outputs of assessment feed back into selection, helping *eQual* to choose better alternatives, thereby generating improved variants. Figure 3 shows *eQual*'s architecture, with the components performing the four steps denoted.

Critically, *eQual*'s inputs are types of artifacts architects must consider and/or create as part of their regular tasks. Software development typically involves modeling of the system's architecture (*eQual*'s step 1), even if informally [71, 83]. 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 architects have to analyze regardless of whether they use *eQual*. 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 amenable to dynamic analysis. Several approaches create such models, including ArchStudio [21], XTEAM [27], PCM [58], and DomainPro [77]. Any of these would suit our purpose. For *eQual*'s implementation, we selected DomainPro [24, 77] because of its simple interface, use of

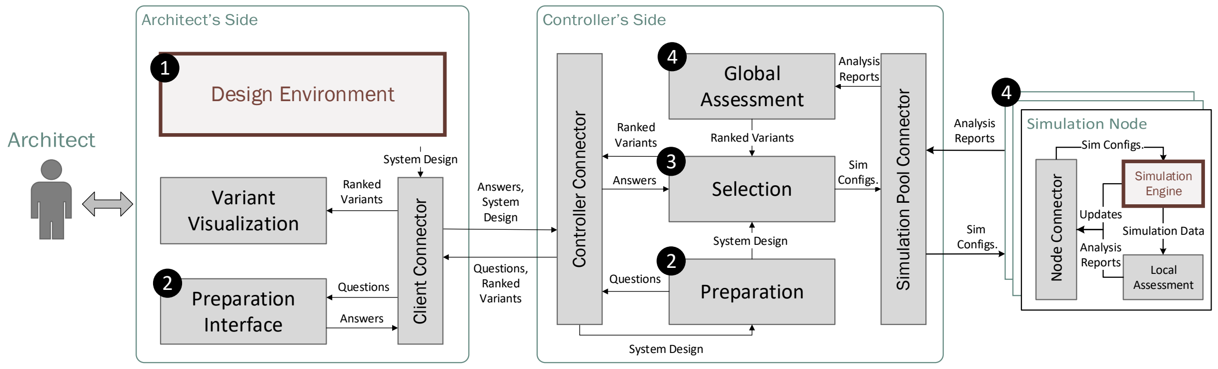

Figure 3: *eQual*'s architecture. The *Design Environment* and *Simulation Engine* components are provided by DomainPro [77].

event-driven simulation, and model-driven architecture (MDA) approach [64] that allows architects to define variation points in their models and try different alternatives (albeit completely manually).

As is common in MDA, a system is designed in DomainPro in two phases. First, an engineer must create a metamodel using DomainPro's editor or reuse a previously defined metamodel. Second, the system is designed by specializing and instantiating elements of this metamodel. A metamodel is a collection of design building blocks relevant to modeling systems of certain kinds or in certain domains that defines a DSL. DomainPro invokes the appropriate model transformation tool (a metainterpreter) to derive the implementation semantics of the DSL types from the metamodel. Subsequently, DomainPro generates a domain-specific model editor, simulation generator (as well as a code generator, which we do not use in *eQual*), all configured with the DSL's custom semantics.

DomainPro provides a built-in metamodel for component-based architectures [62]. The Hadoop model in Figure 1 shows the specialization and instantiation of this metamodel's elements and their depiction in the resulting visual DSL [77]. 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 cloud shape containing the filled-in circles; data-flows are DomainPro `Links` represented with wide arrows; and so on. Figure 1 omits the DomainPro `Parameters` of each modeling element for clarity; several key parameters are shown in Figure 2.

We use this example for illustration. Details of DomainPro and Hadoop's visual DSL from Figure 1 are not necessary for understanding *eQual* and are beyond the scope of this paper.

3.2 Preparation

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

1) Questions Regarding Variation Points

eQual divides variation points into (1) system parameters that are numerical or can be rank-ordered and (2) parameters that are inherently qualitative. For a variation point V from the first category, *eQual* asks architects three questions:

(i) What is V's lower bound?
(ii) What is V's upper bound?
(iii) What is the desired function for exploring V?

The lower and upper bounds capture acceptable ranges of alternatives for each variation point. Exploration functions enable architects to customize how *eQual* samples the specified ranges during design exploration (Sections 3.3, 3.4). For example, in Hadoop, the *Pool Size* variation point's lower bound is 10 and upper bound is 100 (Figure 2). *eQual*'s prototype provides 12 exploration functions: Uniform, Poisson, Gamma, etc.

eQual also allows architects to provide lists of concrete values instead of ranges. This reduces the search space if specific preferences for certain numerical and rank-ordered parameters (category 1) are known *a priori*. This option is *necessary* in the case of qualitative parameters (category 2): Architects must enumerate all relevant alternatives for each qualitative parameter; in turn, *eQual* must select every alternative (Section 3.3) and assess each resulting architectural model for properties of interest (Section 3.4).

For example, let us assume that an architect wishes to explore different RPC mechanisms in Hadoop via three qualitative design parameters associated with DomainPro's `Links`: (1) *Invocation Type*, which can be explicit or implicit; (2) *Synchrony*, which can be synchronous, time-out-synchronous, or aynchronous; and (3) *Delivery Guarantees*, which be at-least-once, at-most-once, exactly-once, or best-effort. In the general case, *eQual* must explore $2 \times 3 \times 4 = 24$ combinations of alternatives for these three parameters *for each combination of alternatives selected for the category 1 variation points.* Note that this growth in the decision space can be stemmed if architects are able to identify specific preferred design choices. For example, a decision to configure Hadoop's `ipc.Client` module to support explicit, asynchronous calls with at-most-once semantics would require that the architects eliminate from the *eQual* model the undesired alternatives for the three parameters, which would reduce the number of combinations of alternatives from 24 to 1.

A prior analysis [78] identified well over 100 design decisions made during part of Hadoop's development. Hadoop's architects considered 2–8 alternatives per variation point. The resulting decision space quickly eclipses human abilities. For example, a minor version involving only 4 new design decisions and 5 alternatives per decision will have over 500 variants. By contrast, the entire burden *eQual* places on architects is to answer $4 \times 3 = 12$ questions about the variation points.

We do not expect architects to be able to answer the above questions right away. *eQual* allows several possibilities: (1) Architects may know the exact answer to a question. (2) They may be able to provide only a partial answer, such as a variation point's lower bound. (3) They may be unable to answer a question, leaving the range of alternatives unbounded.

2) Questions Regarding Non-Functional Properties

eQual's second set of questions deals with the system's NFPs, which are the basis for assessing design alternatives (Section 3.4). The NFPs are determined from system requirements and domain characteristics. For example, in Hadoop, prior work identified four properties [11, 12]: (1) *Reliability* (ratio of tasks that have to be restarted to all tasks), (2) *Machine Utilization*, (3) *Execution Time*, and (4) *Cost* (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 (e.g., Rhapsody [43]), system state is captured at the times of event occurrences. Hence, the output is a set of time-series objects.

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

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

For example, in the case of Hadoop's *Machine Utilization*, the relevant time-series object captures idle capacity of the machine pool in the model discussed above. 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 between 1 and 5, that would treat *Machine Utilization* as more important than, e.g., *Cost* whose coefficient is 1, and less important than *Reliability* whose coefficient is 5). Thus, for the above example of a 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 each for the variation points and properties.

eQual's aim is to *elaborate the information architects already must take into account.* In practice, architects often ignore, accidentally omit, indirectly consider, or incorrectly record information captured by these questions [83]. By (1) strictly bounding the number of questions, (2) consolidating them into one place, and (3) giving them a standard format, *eQual* aims to convert this frequently haphazard process into methodical design. As the architects explore the design alternatives and gain a better understanding of the system, they are 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 inputs of the selection step, whose objective is to explore the space of design variants intelligently and tractably. For example, in Hadoop, this can help engineers explore the effects of non-trivial decisions, such as: What yields better reliability at an acceptable cost, a greater number of less-reliable machines or fewer more-reliable machines?

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 architects during preparation (Section 3.2). We call this initial set *seed variants*, and the process *eQual* uses to pick the seed and later variants *selection strategy*. Seed variants feed into assessment (Section 3.4), where *eQual* comparatively analyzes them. Assessment feeds its ranking of variants back to selection (recall Figure 3), which uses this information to generate an improved set of variants during the next iteration.

Two factors determine selection'a effectiveness: (1) how seed variants are generated and (2) how information from assessment is used to generate subsequent variants. In principle, *eQual* allows any selection strategy. The goal is to enable an architect to control the selection step's number of iterations and generated variants to fit her needs, specific context, and available computational resources.

Our prototype implements two selection strategies based on the genetic evolutionary-algorithm paradigm: random and edge-case seeding. Random seeding chooses seed variants randomly. Edge-case seeding aims to generate variants containing either side of a boundary condition provided to *eQual*. For example, variants in Hadoop would be generated by selecting all lower-bound values from Figure 2 (500, 1, 10, 0.5, 0.5), all upper-bound values (2000, 5, 100, 0.9, 2), and combinations of upper-bound values for some variation points and lower-bound values for the remaining variation points, e.g., (2000, 5, 100, 0.5, 0.5). Note that edge-case seeding is not possible with options that are nominal, i.e., do not have binary or numerical values.

Both strategies quickly prune the space of variants and arrive at good candidate designs (see Section 4). We aim to preserve Pareto optimal [14] solutions at each step as this provides an intuitive way to explore extreme effects of decisions.

3.4 Assessment

To assess a variant's quality, *eQual* dynamically analyzes it via simulation. We chose simulation-based analysis because simulations are representative of a system's real behavior due to their inherent nondeterminism [46]. *eQual* 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 [23]. We thus use DTW.

For each design variant, *eQual* generates a single time-series object for each NFP. 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 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 NFP 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. This may entail transferring hundreds of megabytes of data per variant, and having to redo all of the calculations each time a new variant changes the optimum time-series. Such a solution would be prohibitively costly in scenarios with multiple iterations involving thousands of variants.

To address this problem, we devised the *Bipartite Relative Time-series Assessment* technique (BRTA), which distributes the time-series analysis. As indicated in Figure 3, multiple nodes are tasked with assessing different subsets of variants via simulation. 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 a *local optimum* since the other nodes will perform the same tasks on their 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).

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 as follows:

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

D_g is distance to the global optimum; O_l is the local optimum; D_l is distance to the local optimum; Max_g (Min_g) is the global max (min) value among all time-series of a NFP. We formally prove this formula's correctness in our supplementary material available at https://tinyurl.com/dse-equal.

BRTA's reduction in the amount of transferred data directly facilitates *eQual*'s support for exploring many design alternatives.

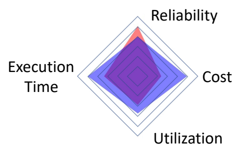

Figure 4: *eQual*'s radar diagram for two candidate Hadoop variants, showing their respective values for four properties of interest.

The updated BRTA summaries include the globally normalized values for each time-series in each variant and are used to rank the variants. To use the D_g values of different NFPs 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.

When multiple variants have comparable qualities, *eQual* also allows architects to visually compare them. Figure 4 shows an example of such a visualization: the architects may use this visualization to decide between the variant that emphasized *Reliability* and the variant that balanced the NFPs more evenly.

4 EVALUATION

We have implemented *eQual* on top of DomainPro [77], resulting in 4.7K C# and 1.0K JavaScript SLoC added to DomainPro. To aid in *eQual*'s evaluation, we also built a utility totaling an additional 1.0K C# and 0.2K MATLAB SLoC, as detailed in Section 4.2.

We conduct analytical and empirical evaluations of *eQual*'s targeted *problem scope*, *usability*, *effectiveness* in finding high-quality solutions, and *scalability*; and a controlled user study further targeting *eQual*'s scope, usability, and effectiveness. We especially, but not exclusively, focus on *eQual*'s comparison to GuideArch, the state-of-the-art approach for exploring early architectural designs.

4.1 *eQual*'s Scope and Usability

Our usability evaluation measures how easy *eQual* is to apply in practice. We present an analytical argument for *eQual*'s usability and results of its empirical evaluation. We also illustrate that *eQual*'s problem scope matches GuideArch's.

1) Analytical Argument

Section 3.2 discussed 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 thus $3 \times (N_v + N_p)$. Recall that the architect has the option of not answering some (or any) of the questions.

As discussed above, our analysis of Hadoop has relied on previously identified four critical NFPs [11, 12]. Prior research suggests that there are usually 4–6 NFPs of interest in a software project, and rarely more than 10 [4]. Moreover, a recent study [78] showed that the number of variation points per Hadoop version ranged between 1 and 12 [78]. 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.

Notably, *eQual* assumes neither that NFPs can be rank-ordered (unlike GuideArch [29] and its predecessor [56]), nor that architects can provide specific fitness functions for them (unlike [79–81]).

2) Empirical Comparison to State-of-the-Art

We modeled Hadoop in GuideArch. We considered other approaches, but found them unsuitable for direct comparison. Beyond approaches already discussed in the Introduction, model-driven solutions [31, 40] target design-space exploration, but require manual specification of model-transformation rules and tackle challenges such as finding the best orchestrations of the rules. Several solutions for optimizing cyber-physical systems [6, 41, 59] rely on simulations but are intimately tied to the underlying domain. For example, AutoFocus [10, 41] targets reactive systems, in which modeling elements are domain-specific abstractions (e.g., electronic control unit) and constraints are linked to the domain's semantics (e.g., traffic-light behavior). As another example, OpenMDAO [35, 59] numerically optimizes designs of multidisciplinary systems such as aircraft, and relies on the availability of mathematical models for properties such as lift, thermodynamics, etc.

We compared *eQual* and GuideArch models in terms of numbers of field entries and time required to complete them. We used Hadoop because the details of the case study published by GuideArch's authors are no longer available. GuideArch helps architects make decisions using fuzzy math [91] to deal with uncertainties about system variation points. GuideArch uses three-point estimates: architects must provide (1) pessimistic, (2) most likely, and (3) optimistic values to describe the effects of their decisions on the system's NFPs. For instance, in the case of Hadoop's *Processing Power*, for each decision (e.g., using machines with 2GHz CPUs) architects have to specify the three values for *Utilization*, *Execution Time*, and *Cost*.

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. Its authors acknowledge that, even for architects who are intimately familiar with a system, it may be challenging to accurately estimate, e.g., the pessimistic value of resource *Utilization* or the most likely system *Reliability*. For this reason, GuideArch's authors recommend that architects obtain this information by analyzing prior data, looking at similar systems, studying manufacturer specifications, and consulting publications [29]. These are nontrivial tasks, likely to rival the modeling effort required by *eQual*.

The specification of NFPs 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 cannot explore ranges.

We highlight an experiment representative of the side-by-side use of the two techniques: 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*—which is allowed by *eQual* but not by GuideArch—we explicitly provided 10, 25, 50, 75, and 100 as the options. The next step was to specify how each decision affects the system's NFPs. 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 NFPs. Overall, it took one of this paper's authors more than four hours to complete over 350 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 options within a variation point). In general, if T_f is the 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 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 Scope and Effectiveness

Most other approaches concerned with design quality (e.g., [5, 21, 27, 58, 77]) focus on single variants and do not explore design-decision spaces (see Section 5 for further details). Prior work [28] has shown that those techniques that aid engineers with arriving at effective designs (e.g., ArchDesigner [1]) underperform GuideArch in the quality of their top-ranked designs. For these reasons, we evaluated *eQual*'s effectiveness by directly comparing it with GuideArch as the leading competing approach. We then separately assessed the quality of *eQual*'s results on systems with known optimal configurations, in the process further highlighting *eQual*'s scope. Our results indicate that *eQual* produces effective designs, of higher quality than competing work.

1) Head-to-Head Comparison with 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 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.). In fact, we observed that GuideArch consistently selects variants with lower machine reliability but higher redundancy than those selected by *eQual*.

We acknowledge that we are intimately familiar with *eQual*. However, the author who performed the analysis has extensive experience with GuideArch. Moreover, we have a good understanding of Hadoop and made every effort to use GuideArch fairly, consulting its authors regularly. Ultimately, 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 NFPs, a non-trivial task regardless of one's familiarity with GuideArch.

2) 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 5. Fitness models describe the NFPs of a system using its variation points and their interactions. These models were obtained by Siegmund et al. [79, 80] and shown to accurately predict the NFPs of highly configurable systems. The 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 models because they are analogous to our objective in *eQual*, despite being applied to systems that are already deployed. Furthermore, the subject systems' resulting decision spaces range from 100K to 1 quadrillion variants, making them attractive for testing *eQual*'s range of applicability.

A fitness model is a function from variants to a fitness measurement $\Omega: C \rightarrow \mathbb{R}$, where fitness can be an aggregation of any measurable NFP that produces interval-scaled data. The model is described as a sum of terms over variation option values. The model's individual terms can have different shapes, such as $n \cdot c(X)$, $n \cdot c(X)^2$ or $n \cdot c(X) \cdot \sqrt{c(Y)}$ [79]. For illustration, a configurable DBMS with options encryption (E), compression (C), page size (P), and database size (D) may have the following fitness model:

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

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 NFP 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 1K C# SLoC and an additional 0.2K MATLAB SLoC to tune and visualize the resulting *eQual* models.

Figures 6 and 7 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 6 compares *eQual*'s two strategies against the solutions yielded by using the default values suggested by the six systems' developers [79]. These results

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 5: Systems used to evaluate *eQual*'s effectiveness. *Var. Points* is number of variation points; *Terms* is number of terms in systems' fitness models; *Size* is number of design-space variants.

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 6 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 7 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. Note 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.

4.3 Scope, Usability, and Effectiveness User Study

We conducted a within-subject controlled experiment with 15 participants using *eQual* and GuideArch to measure (1) whether users were more likely to produce higher-quality designs using GuideArch, *eQual*, or without any tool support, and (2) whether the users preferred using GuideArch, *eQual*, or neither tool.

All participants had industrial experience developing software systems, working for companies such as, Google, Facebook, Samsung, and Cloudera. On average, the participants had 11 years of programming experience (minimum 4, maximum 24) and 6.5 years of software design experience (minimum 1, maximum 15). Additionally, 60% of participants work on software design tasks “somewhat regularly” or “always”. Two-thirds of participants reported that they use models to describe their software “somewhat regularly” or “always”.

Each participant was asked to work with three systems selected from the set in Figure 5: Apache HTTPD (an HTTP server), Berkeley DB C (an embedded database library), and LLVM (a compiler infrastructure). For each system, participants were given a performance objective and variation points for which they had to determine the optimal options. The participants were provided with documents that described each system, its variation points under consideration, and online resources with more information about the system.

Each participant took part in three treatments: using no tool (the control), GuideArch, and *eQual*. The order of treatments was randomized, as was the assignment of treatments to subject systems.

		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 6: Comparison between the two seeding strategies employed by *eQual*, and the quality of the solutions commonly selected by architects (*Default* column, obtained from [79].)

Participants were given one hour to complete the study. They answered questions about each treatment as they progressed through the experiment. At the end, participants were asked to compare the different treatments.

In their assessment of *eQual*, participants were provided with screenshots to describe aspects of the system models that depended on DomainPro [77]. The objective was to minimize the impact on the study's results of DomainPro's details incidental to *eQual* (e.g., metamodeling features). For GuideArch, participants used the online version of the tool [37]. Finally, in the control treatment, they performed the tasks manually.

We compared the top-ranked solution for each treatment submitted by the participants. Results were normalized to determine their distance from the respective global optima, resulting in a quality score between 0 and 1. We compared the distributions of the qualities of the designs produced for the three treatments. The median resulting system quality was 0.872 (mean 0.755) for the GuideArch group, 1.0 (mean 0.999) for *eQual*, and 0.879 (mean 0.719) for control. We applied the Mann-Whitney U Test and found the difference between median system qualities for GuideArch and *eQual* to be statistically significant ($p = 0.00014$) and the effect size large ($r = 0.70$). Likewise, the difference between median system qualities for the control group and *eQual* was statistically significant ($p < 0.00001$) and the effect size large ($r = 0.85$).

Participants reported confidence in their solutions on a Likert scale [53], from Very Unsure (1) to Very Confident (5). *eQual* users had a mean confidence of 4.07, GuideArch users 3.07, and control group 3.14. Student's t-tests pairwise comparing *eQual* vs. GuideArch and *eQual* vs. control group showed statistical significance, with respective p values of 0.03 and 0.02.

Participants also rated two usability aspects of each treatment on a Likert scale. They rated each treatment's required effort, ranging from Very Intensive (1) to Very Easy (5). The mean scores for the *eQual*, GuideArch, and control groups were 4.00, 2.46, and 2.07, respectively. Student's t-tests pairwise comparing *eQual* vs. GuideArch and *eQual* vs. control group showed statistical significance, with respective p values of 0.0028 and 0.00037. Additionally, participants rated the user friendliness of each treatment, ranging from Very Challenging (1) to Very Friendly (5). The mean scores for the *eQual*, GuideArch, and control groups were 4.53, 2.67, and 2.71, respectively. Again, Student's t-tests pairwise comparing *eQual* vs. GuideArch and *eQual* vs. the control group showed statistical significance, with respective p values of 0.00021 and 0.0012.

These results show that participants saw value in using *eQual* and found its interface more intuitive than GuideArch's.

4.4 *eQual*'s Scalability

To evaluate *eQual*'s scalability, we used 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 RAM) as simulation nodes, and a single n1-standard-2 node (2 vCPU and 7.5 GB RAM) as the central controller node. All nodes were located in Google's us-central1-f datacenter. We used the variation points and NFPs described in Section 3.

Number of Nodes. We evaluated *eQual*'s observed speed-up when increasing the number of simulation nodes. We used the general genetic algorithm for 8 generations and the generation size of 256

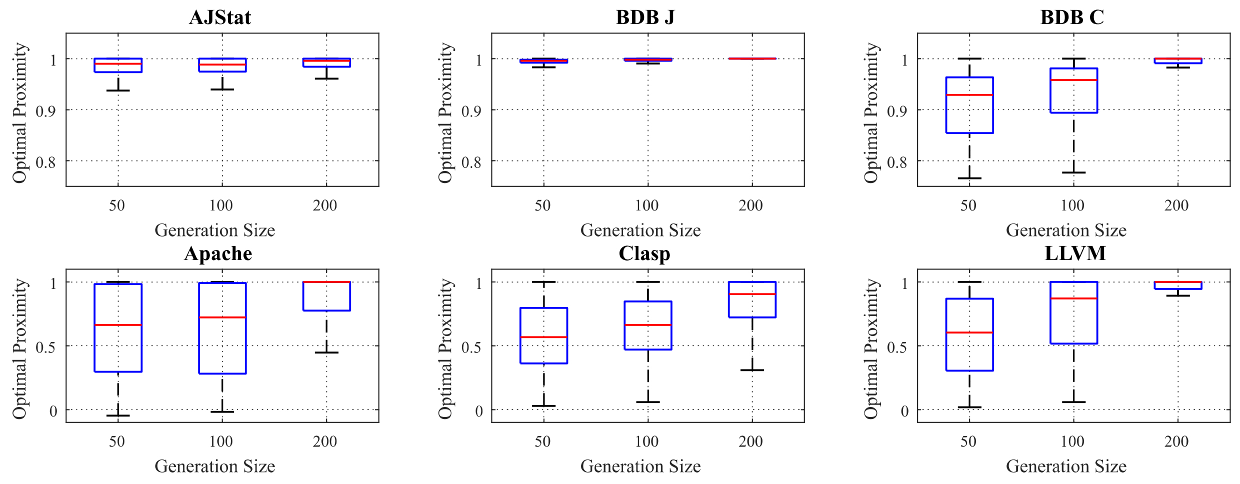

Figure 7: *Optimal Proximity* distribution of the best variant generated by *eQual* using generation sizes of 50, 100, and 200. Each box plot comprises 100 executions of *eQual*'s exploration function using the edge-case strategy.

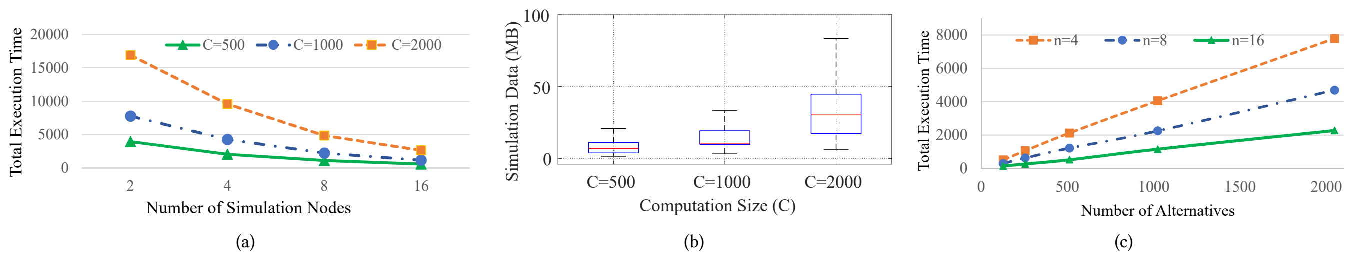

Figure 8: *eQual*'s scalability with number of (a) simulation nodes, (b) events, and (c) variants.

variants, totaling 2,048 variants. We did this with 2, 4, 8, and 16 nodes, and for three values of the *Computation Size* variation point (denoted with *C* in Figure 8). As shown in Figure 8(a), the execution time was inversely proportional to the number of nodes, 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 8(a) consists of 2,048 simulations. Overall, we simulated more than 24,500 design variants.

Number of Events. We also measured the impact of increasing the number of events generated during a simulation. This is indicative of how 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 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 8(b) 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* (Figure 8(a)), this is indicative of *eQual*'s scalability as numbers of simulation events grow.

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

4.5 Threats to Validity

While *eQual*'s evaluation finds it is easy to use, scales well, has a small footprint, and finds accurate solutions, we take several steps to mitigate possible threats to our work's validity.

The controlled experiment had two potential validity threats that arose from practical considerations. We believe these do not impact the design quality or usability results. The first consideration is the participants' use of existing ground-truth models, rather than creating their own. This was necessary to enable objective comparison between the user-generated solutions. Moreover, these models were produced by a third party [79, 80]. The second threat stemmed from having to execute the study in a limited amount of time. Recall from Section 4.3 that we relied on screenshots of previously developed architectural models to explain the full scope of *eQual* (notably its DomainPro modeling substrate). Furthermore, we summarized subject systems to enable participants to understand the salient aspects of each system relatively quickly. Finally, we asked the participants to consider only a single NFP. We believe that the cumulative effect of thus simplified scenarios may have worked in favor of the control groups. At the same time, we posit that the net effect on *eQual* and GuideArch was mostly neutral: the reduction in modeling effort, which favored *eQual*, is balanced out by providing a system "digest" and focusing on a single NFP, favoring GuideArch.

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.) [82, 84], and the bedrock of our assessment models, DTW, is so prevalent that it is "difficult to overstate its ubiquity" [23, 70]. DTW also subsumes Euclidean distance [23] as a special case [70], which increases *eQual*'s range of applicability. The threat to the **external validity** of our work is mitigated by the incorporation of an MDA-based approach (namely, DomainPro). MDA solutions have been shown to be sufficiently robust and scalable, and are widely used in research and industry [42, 47].

5 RELATED WORK

eQual builds on a large body or work that has explored support for making design decisions via static or dynamic analysis of software models [45, 69]. Static analyses (e.g., [8, 34, 60]) tend to require architects to develop complex mathematical models, imposing steep learning curves, modeling effort, and limits on system scalability. Depending on the mathematical models they rely on (e.g., Markov chains [33], event calculi [48], or queueing networks [9]), these techniques are confined to specific kinds of software system models [2].

While they come with shortcomings of their own (e.g., false negatives, long execution times), dynamic analysis techniques—i.e., architectural model *simulations* [27, 55]—are more capable of capturing the nondeterminism reflective of reality [46]. Despite notable efforts, especially in domains with well understood properties (e.g., stream-based systems [22], reactive hybrid systems [10, 41, 85], numerical optimization of aircraft designs [35, 59]) simulations of software architectural models [57, 89] have not been employed as widely as static analyses [2], for at least four reasons. First, creating simulatable system design models is *difficult* [27]. Second, running simulations is time consuming, mandating that *scalability* be explicitly addressed [68]. Third, quantitative assessment of variants is a *complex* computational problem because of the involved tradeoffs [61]. Finally, analysis and understanding of *massive datasets* may be necessary to assess system behavior. *eQual* has built-in features to explicitly deal with each of these problems.

Rule-based approaches identify problems in a software model and rules to repair them. MOSES uses stepwise refinement and simulation for performance analysis [19]. ArchE helps meet the quality requirements during the design phase by supporting modifiability and performance analysis [60]. DeepCompass relies on a Pareto analysis to resolve conflicting goals of performance and cost between different embedded-system architecture candidates [7]. PUMA facilitates communication between systems designed in UML and NFP prediction tools [90]. FORMULA aims to reduce the design-space size by removing candidates based on a user-defined notion of design equivalence [44]. Unlike *eQual*, each of these approaches is limited by its predefined rules and cannot explore the complete design space.

Metaheuristic approaches treat architecture improvement as an optimization problem. ArcheOpterix [2], PerOpteryx [49], and DeSi [56] use evolutionary algorithms to optimize system deployment with respect to quality criteria. PerOpteryx offers predefined degrees of freedom for optimizing deployment. PerOpteryx's optimization strategy can be incorporated into *eQual*, with allowances for *eQual*'s broader scope. AQOSA supports modeling based on AADL and performance analysis tools, and evaluates design alternatives based on cost, reliability, and performance [52]. SASSY targets SOAs, selecting services and a pattern application to fulfill quality requirements [63]. Metaheuristic simulation has been used to reconfigure a legacy application for the cloud [32]. Linear programming has been used to find minimum-cost configurations on the cloud [3]. DESERT explores design alternatives by modeling variations in a tree and using Boolean constraints to eliminate infeasible solutions [26]. DESERT-FD automates constraint generation and design exploration [26]. GDSE uses meta-programming of domain-specific design exploration problems and expresses constraints for solvers to generate solutions [72]. *eQual* uses metaheuristic search to find solutions within large design spaces, but focuses on NFPs and lowering the burden on architects.

Software Product Lines (SPLs) support configuring software artifacts for a set of requirements [18, 39, 74, 81]. Unlike SPLs, *eQual* neither adds nor removes features in a product. SPLs can use genetic algorithms to optimize feature selection [38], but this requires developers to create objective functions to measure variants' fitness. Optimizing configurable systems, despite being aimed at already deployed systems, has clear relations to *eQual*. Among these techniques we used the studies by Siegmund et al. [79, 80] to evaluate the effectiveness of *eQual*. Oh et al. [67], and Sayyad et al. [73] have devised techniques to more efficiently explore the space of system configurations, which can complement *eQual*'s exploration strategies. SPLs can also be modeled as a combination of a core model, representing one variant (product) and a set of Δ-models for the differences with other variants [75]. Δ-modeling is a way to model feature variability, whereas *eQual* employs search-based strategies on an underlying architectural model of a system to explore its design variants.

6 CONTRIBUTIONS

Our work provides an important step in narrowing the chasm between the needed and available support for making and evaluating early architectural design decisions. Our approach, *eQual*, guides architects in making informed choices, by quantifying the consequences of their decisions throughout the design process. Critically, *eQual* provides structure and automated support to the architects' *already existing* tasks. *eQual* does so while being able to navigate efficiently through massive design spaces. *eQual* is able to *simultaneously* match or better the state-of-the-art in terms of four key dimensions: problem scope, usability, effectiveness, and scalability.

While our results show promise, further work is needed to improve *eQual*'s practical effectiveness. Thus far, we have assumed that architects know the relative importance of NFPs in their systems. Our goal is to actively guide architects in the identification of design hot-spots and help their understanding of the NFPs' relative importance. Moreover, combining *eQual* with a software architecture recovery technique will extend its applicability to existing systems with legacy architectures. *eQual*'s application to systems with known fitness models [79, 80] supports this idea's viability.

REFERENCES

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

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

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

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

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

[6] Torsten Blochwitz et al. 2011. The functional mockup interface for tool independent exchange of simulation models. In *International Modelica Conference*. Dresden, Germany, 105–114. https://doi.org/10.3384/ecp11063105

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

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

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

[10] Manfred Broy et al. 2008. Service-Oriented Modeling of CoCoME with Focus and AutoFocus. In *The Common Component Modeling Example*, Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Raffaela Plášil (Eds.). Springer-Verlag, 177–206. https://doi.org/10.1007/978-3-540-85289-6_8

[11] Yuriy Brun, George Edwards, Jae young Bang, and Nenad Medvidovic. 2011. Smart redundancy for distributed computation. In *International Conference on Distributed Computing Systems (ICDCS)* (20–24). Minneapolis, MN, USA, 665–676. https://doi.org/10.1109/ICDCS.2011.25

[12] Yuriy Brun, Jae young Bang, George Edwards, and Nenad Medvidovic. 2015. Self-Adapting Reliability in Distributed Software Systems. *IEEE Transactions on Software Engineering (TSE)* 41, 8 (August 2015), 764–780. https://doi.org/10.1109/TSE.2015.2412134

[13] Saheed A. Busari and Emmanuel Letier. 2017. RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. Buenos Aires, Argentina, 552–562. https://doi.org/10.1109/ICSE.2017.57

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

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

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

[17] Paul Clements, Rick Kazman, and Mark Klein. 2001. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional.

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

[19] Vittorio Cortellessa, Pierluigi Pierini, Romina Spalazzese, and Alessio Vianale. 2008. MOSES: MOdeling Software and platform architEcture in UML 2 for Simulation-based performance analysis. In *Quality of Software Architectures. Models and Architectures*. Springer, 86–102.

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

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

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

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

[24] Christoph Dorn, George Edwards, and Nenad Medvidovic. 2012. 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"*. 362–379.

[25] Gerald S Doyle. 2011. *A methodology for making early comparative architecture performance evaluations*. Ph.D. Dissertation.

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

[27] George Edwards, Yuriy Brun, and Nenad Medvidovic. 2012. 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*. 161–170.

[28] Naeem Esfahani. 2014. *Management of uncertainty in self-adaptive software*. Ph.D. Dissertation. George Mason University.

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

[30] Mahdi Fahmideh and Ghassan Beydoun. 2019". Big data analytics architecture design — An application in manufacturing systems. *Computers & Industrial Engineering* 128 (2019"), 948 – 963. https://doi.org/10.1016/j.cie.2018.08.004

[31] Martin Fleck, Javier Troya, and Manuel Wimmer. 2015. Marrying search-based optimization and model transformation technology. In *First North American Search-Based Software Engineering Symposium (NasBASE)*. Elsevier, 1–16.

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

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

[34] Swapna S Gokhale. 2004. Software application design based on architecture, reliability and cost. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, Vol. 2.

[35] Justin Gray, Kenneth Moore, and Bret Naylor. 2010. OpenMDAO: An Open Source Framework for Multidisciplinary Analysis and Optimization. In *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*. https://doi.org/10.2514/6.2010-9101

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

[37] GuideArch V1.0 2012. GuideArch V1.0. http://mason.gmu.edu/~nesfaha2/Projects/GuideArch/.

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

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

[40] Ábel Hegedüs, Ákos Horváth, and Dániel Varró. 2015. A model-driven framework for guided design space exploration. *Automated Software Engineering* 22, 3 (2015), 399–436.

[41] Florian Hölzl and Martin Feilkas. 2010. 13 AutoFocus 3 — A Scientific Tool Prototype for Model-Based Development of Component-Based, Reactive, Distributed Systems. In *Model-Based Engineering of Embedded Real-Time Systems*. Springer Berlin Heidelberg, Dagstuhl, Germany, 317–322. https://doi.org/10.1007/978-3-642-16277-0_13

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

[43] IBM. [n.d.]. IBM Rationale Rhapsody. http://www-03.ibm.com/software/products/en/ratirhapfami.

[44] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. 2011. An Approach for Effective Design Space Exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, Radu Calinescu and Ethan Jackson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–54.

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

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

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

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

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

[50] Emmanuel Letier, David Stefan, and Earl T. Barr. 2014. Uncertainty, Risk, and Information Value in Software Requirements and Architecture. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. Hyderabad, India, 883–894. https://doi.org/10.1145/2568225.2568239

[51] Daniel R. Levinson. 2014. 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.

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

[53] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[54] FMS Luke Chung. 2013. Healthcare.gov is a Technological Disaster. http://goo.gl/8B1fcN.

[55] Jiefei Ma, Franck Le, Alessandra Russo, and Jorge Lobo. 2016. Declarative Framework for Specification, Simulation and Analysis of Distributed Applications. *IEEE Transactions on Knowledge and Data Engineering* 28, 6 (2016), 1489–1502.

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

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

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

[59] Joaquim R.R.A. Martins and Andrew B. Lambe. 2013. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal* 51, 9 (2013).

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

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

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

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

[64] Joaquin Miller, Jishnu Mukerji, M Belaunde, et al. 2003. MDA guide. *Object Management Group* (2003).

[65] Tim Mullaney. 2013. Demand overwhelmed HealthCare.gov. http://goo.gl/k3o4Rg.

[66] Joost Noppen, Pim van den Broek, and Mehmet Akşit. 2008. Software development with imperfect information. *Soft computing* 12, 1 (2008), 3.

[67] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-optimal Configurations in Product Lines by Random Sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. ACM, New York, NY, USA, 61–71. https://doi.org/10.1145/3106237.3106273

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

[69] Pasqualina Potena. 2007. 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)*. 583–586.

[70] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2013. 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 (2013), 10.

[71] M. P. Robillard and N. Medvidovic. 2016. Disseminating Architectural Knowledge on Open-Source Projects: A Case Study of the Book "Architecture of Open-Source Applications". In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 476–487. https://doi.org/10.1145/2884781.2884792

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

[73] A. S. Sayyad et al. 2013. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 465–474. https://doi.org/10.1109/ASE.2013.6693104

[74] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. 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)*. 492–501.

[75] Ina Schaefer. 2010. Variability Modelling for Model-Driven Development of Software Product Lines. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, Vol. 10. Linz, Austria, 85–92.

[76] Ali Sedaghatbaf and Mohammad Abdollahi Azgomi. 2019. SQME: A framework for modeling and evaluation of software architecture quality attributes. *Software & Systems Modeling* 18, 4 (Aug. 2019), 2609–2632. https://doi.org/10.1007/s10270-018-0684-3

[77] Arman Shahbazian, George Edwards, and Nenad Medvidovic. 2016. An end-to-end domain specific modeling and analysis platform. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*. 8–12.

[78] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. 2018. Recovering Architectural Design Decisions. In *IEEE International Conference on Software Architecture (ICSA)*.

[79] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-influence Models for Highly Configurable Systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. ACM, New York, NY, USA, 284–294. https://doi.org/10.1145/2786805.2786845

[80] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. RosenmÃijller, and G. Saake. 2012. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*. 167–177. https://doi.org/10.1109/ICSE.2012.6227196

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

[82] Ghanem Soltana, Nicolas Sannier, Mehrdad Sabetzadeh, and Lionel C Briand. 2015. 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*. 70–79.

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

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

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

[86] United States Government Accountability Office. 2015. Report to Congressional Requester, GAO-15-238. http://www.gao.gov/assets/670/668834.pdf.

[87] US Centers for Medicare and Medicaid Services. 2013. McKinsey and Co. Presentation on Health Care Law. http://goo.gl/Nns9mr.

[88] US Department of Health and Human Services. 2013. HealthCare.gov Progress and Performance Report. http://goo.gl/XJRC7Q.

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

[90] Murray Woodside, Dorina C Petriu, Dorin B Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer. 2005. Performance by unified model analysis (PUMA). In *Proceedings of the 5th International Workshop on Software and Performance*. 1–12.

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