Dynamic Evaluation of Microservice Granularity Adaptation

SARA HASSAN, Birmingham City University, UK
RAMI BAHSOON and LEANDRO MINKU, University of Birmingham, UK
NOUR ALI, Brunel University London, UK

Microservices have gained acceptance in software industries as an emerging architectural style for autonomic, scalable, and more reliable computing. Among the critical microservice architecture design decisions is when to adapt the granularity of a microservice architecture by merging/decomposing microservices. No existing work investigates the following question: how can we reason about the trade-off between predicted benefits and cost of pursuing microservice granularity adaptation under uncertainty? To address this question, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem. We propose a novel evaluation process for dynamically evaluating granularity adaptation design decisions under uncertainty. Our process is based on a novel combination of real options and the concept of Bayesian surprises. We show the benefits of our evaluation process by comparing it to four representative industrial microservice runtime monitoring tools which can be used for retrospective evaluation for granularity adaptation decisions. Our comparison shows that our process can supersede and/or complement these tools. We implement a microservice application — Filmflix — using Amazon Web Service (AWS) Lambda and use this implementation as a case study to show the unique benefit of our process compared to traditional application of real options analysis.

CCS Concepts: • **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Software and its engineering** → **Extra-functional properties**; • **General and reference** → **Design**; **Evaluation**.

Additional Key Words and Phrases: microservices, granularity, decisions, runtime, software economics

1 INTRODUCTION

Several industries have been leveraging microservices [14] — a more fine-grained and autonomic form of services — to introduce added value to the software architecture. The added value can come in the form of the architecture's ability to cope with operation, maintenance, and evolution uncertainties aiming for cost-effective quality of service (QoS) provision to end users.

One of the critical challenges related to the transition to a microservice architecture design is reasoning about its suitable granularity level. A granularity level determines "the service size and the scope of functionality a service exposes [36, p.426]." Granularity adaptation entails merging or decomposing functionalities across microservices; thereby, moving to a finer or more coarse grained granularity. This problem is critical as software architects may not be able to justify the value added of splitting decisions and its consequences on qualities. Architects may learn about the suitable granularity levels, along the architecture's lifetime, rather than at the beginning of transiting to microservices [14].

The common industrial practice is to reason about microservice granularity adaptation through retrospective analysis, using metrics and operations logs recorded at runtime by microservice monitoring tools [28, 40, 67]. Retrospective reasoning can lead to cases where by time the decision to adapt granularity is made, the architecture may have already suffered the negative consequences of an unsuitable granularity adaptation. *We address the general lack of evaluation processes which can aid architects in quantifying the potential added value of granularity adaptation and the fluctuation*

Authors' addresses: Sara Hassan, Birmingham City University, Millennium Point, Birmingham, UK, B4 7XG, Sara.Hassan@bcu.ac.uk; Rami Bahsoon, r.bahsoon@cs.bham.ac.uk; Leandro Minku, l.l.Minku@cs.bham.ac.uk, University of Birmingham, Edgbaston, Birmingham, UK, B15 2TT; Nour Ali, Brunel University London, Kingston Lane, London, UK, UB8 3PH, nour.ali@brunel.ac.uk.

of that added value over time; adaptation decisions can be then made as soon as the potential added value of the current granularity level becomes inadequate.

Consider a running microservice application with a functionality and scale of Netflix — called NetWatch. It receives around one billion streaming requests everyday [27]. Consider that NetWatch's performance is being monitored at runtime using AWS Cloudwatch indicating a deterioration in NetWatch performance. One possible resolution is for the architect to pursue granularity adaptation assuming that it can improve NetWatch's performance. However, it might be the case that keeping the current granularity level may add more value to NetWatch than the value of immediate performance improvement resulting from pursuing granularity adaptation. The added value is uncertain and can fluctuate with changes in requirements and the environments ; the architect needs to factor uncertainty and its sources is her/his predictions when valuing the decisions to pursue granularity adaptation (or not). Sources of uncertainty include fluctuations in the number of end users, messages exchanged across microservices etc.

Granularity adaptation at runtime can be viewed as an exercise that can introduce an added value under uncertainty. Examples of added value include improvement in load balancing and preventing bottlenecks on microservices that may experience high loads. A counter argument may make a case for keeping the current granularity level unchanged if it proves to have more added value compared to the value which can be introduced by granularity adaptation. This paper aims at answering the following question: *"how can the potential added value of granularity adaptation be evaluated dynamically to determine when is it worth pursuing adaptation?"* The central theme of this paper is that *granularity adaptation is only worthwhile if it will introduce added value to the architecture.* The added value which can be enabled by granularity adaptation is dynamic. It varies depending on usage scenarios and context, fluctuations in workload and level of QoS provision. Henceforth, the potential added value of granularity adaptation is dynamic and can be best tracked and analysed at runtime based on the observed runtime behaviour of the microservice architecture.

The novel contribution of this paper is two-fold. First, we provide a novel formulation of the decision problem to pursue granularity adaptation as a real options problem. Second, we contribute to *a novel evaluation process for dynamically evaluating the granularity adaptation design decisions under uncertainty at runtime.* Our process evaluates potential added value through a novel hybrid combination of binomial real options theory [66] and the concept of Bayesian surprises [9]; it is customised for the context of microservice granularity adaptation. Our process dynamically determines at runtime if it would be worth adapting granularity from the perspective of introducing added value [10]. Unlike classical design time of binomial options analysis in software architecture [48, 64], and [63], we apply the analysis at runtime and we extend the formulation to cater for runtime changes in the potential added value. Bayesian surprises quantify the divergence between uncertain beliefs (claims [9]) about a system's runtime behaviour prior and posterior to observing runtime evidence [9]. Our evaluation process uses Bayesian surprises [9] to enable binomial real options analysis to do real time updates of added value in response to runtime evidence variable values (e.g., workload variations).

On the methodological level, we use our AWS Lambda implementation of an online movie review microservice application — called Filmflix — to demonstrate how our evaluation process goes beyond traditional application of real options. Initially, we compare the usage of our novel evaluation process to the usage of four industrial microservice monitoring tools in retrospective reasoning about granularity adaptation decisions. The comparison takes two forms: (1) comparing the output of our process against that of AWS Cloudwatch and, (2) discussing the differences between the features available in three other microservice monitoring tools and those provided by our process. This shows the unique benefit of our process: linking microservice granularity adaptation to its potential added value under uncertainty. Then we use our demonstration to show how our contribution is unique in revealing dynamic trends related to potential

added value of granularity adaptation at runtime in response to variations in workload. Finally, we discuss the ability of our process to accommodate additional input to it.

Section 2 discusses related work. Section 3 presents our novel formulation of the granularity adaptation decision problem as a real options problem. Section 4 describes traditional binomial real options analysis [66], Bayesian surprises [9] then our novel evaluation process. Section 5 reports on the evaluation of the approach using Filmflix. Section 6 discusses threats to validity and Section 7 concludes with short- and long-term research directions.

2 RELATED WORK

In Section 2.1, we describe representative approaches for reasoning about microservice granularity adaptation decisions and how they compare to our work. In Section 2.2, we compare value-driven decision-making approaches in software engineering to our novel usage of real options analysis. We focus particularly on value-driven approaches since they are the closest to the central theme of this paper. In Section 2.3, we summarise previous usages of real options analysis the field of software engineering to highlight the novelty of applying real options analysis in the microservices context. In Section 2.4, we compare our evaluation process to relevant state-of-the-art runtime adaptation decision support techniques to highlight the novelty of our contribution's role.

2.1 Reasoning about Microservice Granularity Adaptation

The common industrial practice is to reason about microservice granularity adaptation through retrospective analysis, using metrics and operations logs recorded at runtime by microservice monitoring tools (e.g., Amazon Web Service (AWS) Cloudwatch [67], X-Pack [40], IBM Bluemix [28], and Riemann). For example, the architect could retrospectively analyse the network latency and fault rate logs using fault monitoring and diagnostic tools then determine which problems can be resolved by adapting microservice granularity levels. As an example, if the logs indicate latency across the microservices, the architects might reason that this latency can limit the architecture's ability to scale and/or to meet its performance requirements. They can then decide to adapt the granularity. Retrospective reasoning does not quantify the potential added value of granularity adaptation decisions (e.g., [21] [38]) and/or consider the uncertainty related to the suitability of these decisions (e.g., [56]). Both aforementioned gaps are targeted and addressed by our contributions in this paper.

The common practice in developing microservices is to use design patterns; the practice can implicitly or explicitly inform reasoning about granularity. Examples of documented guidelines to support this reasoning include [21, 38, 44, 52]). Moreover, microservice-specific design patterns and best practices that can influence microservice granularity adaptation decisions have been proposed (e.g., [26, 35, 46, 47, 65, 68], and [50]). Our search yielded around 30 microservice-specific approaches, some of which have been applied in industry. Among the factors considered when reasoning about granularity: performance, reliability, scalability, maintainability, and complexity. The use of guidelines, patterns, and best practices could help achieve a reasonable granularity, but they do not objectively evaluate for the *added value* of adopting any of these patterns. Our process can complement these guidelines by dynamically evaluating the added value related to each candidate patterns along its objective performance.

2.2 Value-Driven Decision-Making Approaches

Perhaps the closest to our formulation of the granularity adaptation problem are the approaches presented in [48, 63, 64] where refactoring the architecture is regarded a value-bearing investment. However, these approaches are only applied

statically at design time. In our work, we go beyond static evaluation at design time and apply it in a runtime context to track the added value of granularity adaptation in dynamic microservice environment.

Similar to our approach, net present value captures the notion of enabled opportunities in terms of direct cash flow generated [61]. However, net present value does not capture uncertainty when evaluating opportunity cost/options [16]. In this paper, we tailor traditional real options analysis to make it a dynamic rather than static value-driven approach. Our approach captures dynamics in the operating environments and in QoS fluctuations and how these dynamics can impact real option values.

In [4], a cost benefit analysis method (CBAM) is proposed as a generic architecture evaluation method which utilises techniques in decision analysis, optimisation, and statistics to evaluate architectural design decisions. However, CBAM does not dynamically track and update the added value of architectural decisions.

2.3 Previous Usages of Real Options Analysis

Real options theory has been used in many fields including but not limited to: systems design and engineering [8], software refactoring [63] and COTS-centric development [18], Extreme Programming [17], cloud computing and technical debt [2]. Our work however is the first to apply real options analysis in the context of microservice architectures.

2.4 Runtime Adaptation Support for Service-oriented and/or Microservice Architectures

In terms of functionality, the closest we have encountered to our evaluation process is the ASTRO-CAptEvo orchestration framework [33]. It is a runtime framework that allows partial definition of business processes for service-based systems at design-time and their subsequent refinement at runtime. Similar to our contribution, the framework takes a runtime approach to decision-making. However, the decision problem targeted by ASTRO-CAptEvo is service composition rather than microservice granularity and value-driven decision-making.

More specific to microservices is the MicroADS approach in [34]. This work takes an automated approach to optimise the performance and scalability of the microservice architecture based on a given runtime workload. Using workload to inform decision-making is similar to our use of runtime variables to justify granularity adaptation. However, the MicroADS does not analyse the value-driven ramifications of optimising performance and scalability. In fact, we envision MicroADs can benefit from enriching its reasoning with our proposed evaluation process.

Looking at experiences from microservice adopters, [39] promotes the use of service choreography rather than orchestration for Java microservice applications. In particular, choreography is where each system understands how to react to changes in the environment rather than employing the traditional business process of following orders. Even though this approach takes a runtime approach just like our contribution, our contribution is proactive rather than reactive. In particular, our use of real option values is a more cautious approach that promotes granularity adaptation to enable added value that can be exploited in the future once an opportunity appears. We envision that [39] can be leveraged with insights from our evaluation process to allow for proactive value-driven choreography of microservices.

An example of orchestration for large microservice adopters is the Netflix application programming interface (API) [51, 54] which exposes coarse grained APIs by composing fine-grained functionality provided by the microservices. However, Netflix API reasons about service composition in a technical rather than value-driven manner. Similar to the above approaches, the Netflix API experience can be enhanced with a value-driven dimension by utilising the output of our evaluation process as a trigger for adapting the granularity of the exposed APIs.

3 PROBLEM FORMULATION

In this section, we formulate the problem of evaluating the worth of pursuing granularity adaptation as a real options problem. A real option is an investment decision that can be taken under uncertainty as a right but not with no obligation to [45]. In the context of software architecture, different types of real options have been utilised including the option to switch, defer, explore, expand or stage investments [48] and [63]. In our case, the real option is enabled by microservice granularity adaptation and hence granularity adaptation can be mapped to a real option problem as follows:

(1) Granularity adaptation can be viewed as an architectural decision, that if made, may enable real options that enhance the system utility. (e.g., MovieReview performance). The real options enabled by granularity adaptation give architects the right without a symmetric obligation to reap their benefit(s).
(2) The value of the real options enabled by granularity adaptation is uncertain due to fluctuations in the microservice runtime environment (**e.g.,** in workload volumes).
(3) Uncertainty regarding the value of architectural potentials can be updated given runtime observations of the microservice architecture and its runtime environment.

Answering the question of when is it worth pursuing granularity adaptation, we probe for an answer for:

- **RQ1:** What is the potential added value of pursuing granularity adaptation (related to the first point above)?
- **RQ2:** How can uncertainty regarding this potential added value be captured (related to the second point above)?
- **RQ3:** How can this uncertainty be updated in the face of runtime observation of the microservice architecture and its running environment (related to the third point above)?

4 DYNAMIC EVALUATION OF MICROSERVICE GRANULARITY ADAPTATION

In this section, we explain how our process dynamically evaluates at runtime the added value of pursuing granularity adaptation versus keeping the current granularity level unchanged. Our process is iterative as it evaluates the added values then suggests pursuing granularity adaptation or not at the end of each iteration. Section 4.1 describes the Filmflix architecture, our running example used in the following subsections. Sections 4.2 and 4.3 explain our utilisation of traditional binomial real options analysis [66] and Bayesian surprises [9]. Section 4.4 describes our proposed evaluation process which is a novel combination of the aforementioned theories customised for the context of microservice granularity adaptation.

4.1 Filmflix: A Running Example and Evaluation Case

Filmflix is a hypothetical online movie review microservice application inspired by the functionality and scale of Netflix-amongst the largest microservice adopters. Its initial architecture contains three microservices: (1) ReviewRegulation, implementing the regulation of movie reviews, (2) ReviewUpload, managing the user input requirements when uploading a review and, (3) MovieReview, capturing input from the user through a user interface. We consider that the move to microservices is driven by enhancing the performance of Filmflix where the expected runtime workload (number of reviews sent to Filmflix) is high. We consider the invocation duration and the length of each review received (measured by number of sentences per review) as the runtime variables that the Filmflix's utility is sensitive to. Our contribution utilises these variables to inform runtime decisions about whether the Filmflix architecture should be adapted or not. It is worth noting that choosing the specific granularity adaptation strategy (e.g., merging ReviewRegulation and ReviewUpload or decomposing ReviewRegulation into two microservices) is beyond the scope of our contribution.

4.2 Utilising Binomial Real Options Analysis

There are several models for evaluating real options (e.g., Black and Scholes [13], Monte-Carlo methods [23], and finite difference methods [60]). Among the examples of real options is that of Baldwin and Clark's theory [8]; it studies the real options embedded with a modular software architecture design. The theory defines a model for reasoning about the value added to a base system by modularising it. The theory originated from studying the impact of modularised design on structure of a computing industry. The study revealed that "modularity in computer designs caused the industry to evolve from its initial concentrated structure to a highly dispersed structure. Modularity allows design tasks to be divided among groups that can work independently and do not have to be part of the same firm." Consequently, this theory aims to study how modularity can be attained in software architectures, how this can enable modular design evolution (i.e., create real options) and how economic incentives in each module can be exercised autonomously (i.e., exercising the real options). A particularly relevant type of real options to our case is the call option — it is related to buying a stock, bond, commodity or other instrument at an exercise price at any point within a specific time period before the option expires (according to the American view of options). Baldwin and Clark's theory aims to map and quantify the option value of modular software architecture designs. This theory has been applied in multiple complex system contexts (e.g., [57, 59, 62, 69], and [41]).

In the context of Baldwin and Clark's theory, modularising a design entails using operators to change the design of an architecture into modules to embed real options. Microservices granularity adaptation can be seen as a generalisation of the modularisation exercise. Granularity adaptation has been specifically used in the context of microservices to mean merging or splitting functionalities into microservices according to a systematic granularity adaptation strategy [26].

Though granularity adaptation and modularisation exhibit resemblance, there is fundamental differences between the two because of : 1) granularity adaptation takes place in a relatively more dynamic environment compared to modularisation and, 2) modularisation is driven by modularity enhancement and maintainability, while granularity adaptation is often driven by enhancing other microservitization utilities (e.g., autonomy, replaceability, and/or QoS provision measured in terms of performance). However, they both agree on the notion of encapsulating functionalities within boundaries according to a particular driver. We use inspiration from the theory of Baldwin and Clark along two dimensions: 1) we formulate the granularity adaptation decision problem as a real option problem and, 2) we evaluate the added value of real options related to pursuing granularity adaptation. However, our approach to evaluation is different from Baldwin and Clark: firstly, our work is the first to transit binomial real options analysis to runtime (addressing question 3 in Section 3) as opposed to static design-time traditional usage of this theory. Bayesian surprises [9] enable binomial real options analysis to update added value predictions at runtime in response to runtime evidence variable values (e.g., workload variations). Secondly, granularity adaptation can be evaluated relative to the drivers of the exercise and to the added value of the architectural utility; the traditional usage of Baldwin and Clark's theory focuses on modularity as the architectural utility of concern. Additionally, our choice of binomial real options analysis is due to the flexibility it provides in using the architects' estimates for potential rises or falls in the value of the architectural utility over time given a certain granularity level. Traditional use of binomial real options in software engineering (e.g., [48]) relies on the architects' expectations or publicly available cross-company data to inform real option evaluation. These sources can provide insights regarding potential improvement of architectural utilities due to an investment in the architecture. Our initial value capture is consistent with classical work where we use utility trees. Unlike the classics, we update these values at runtime as explained in Section 4.4.

Suppose that the architect is interested in valuing two real options related to adapting and not adapting the granularity of the initial Filmflix architecture. In our work, we use two binomial trees. One binomial tree is used to evaluate the Real Option Value (ROV) of real options that can be enabled if the decision is made to pursue granularity adaptation. An example is the real option to improve the performance of Filmflix by minimising invocation duration. Similarly, another binomial tree is used to evaluate the ROV retained by not adapting granularity. In the following three paragraphs, we use Figures 1 and 2 to explain different parts of constructing binomial trees.

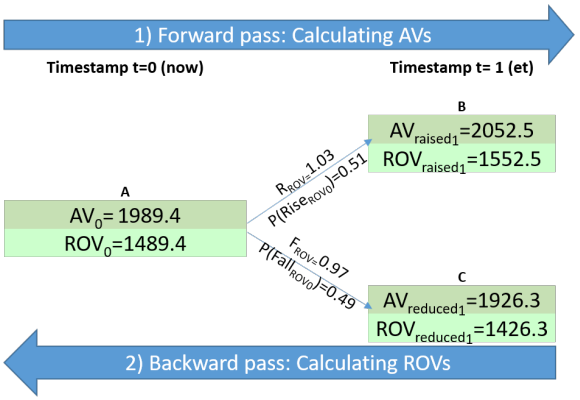


Fig. 1. Example traditional binomial real options analysis tree calculating the Architectural Values (AVs) (e.g., the overall value of a microservice architecture where the granularity is adapted by merging microservices together to improve the application's invocation duration) and real option values (ROVs) for a single real option embedded in the architecture and it is exercised at timestamp 1. The ROV rises at rate R_{ROV} with probability $P(Rise_{ROV_0})$ or falls at rate F_{ROV} with probability $P(Fall_{ROV_0})$, an example of a binomial tree

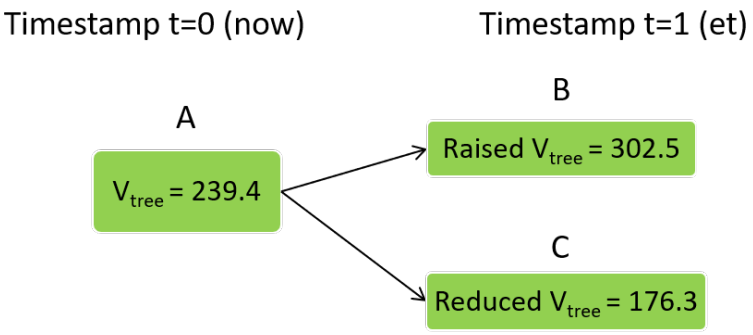


Fig. 2. Example utility tree of an architecture; cells A,B, and C have utility values (V_{Tree}) (e.g., the utility resulting from improving the invocation duration of an architecture by merging microservices together) used to calculate AVs of the respective cells A,B, and C in Figure 1

The bottom parts of the cells (*ROVs*) in Figure 1 show the real option values at different timestamps. The timestamp t=0 is when the decision of adapting the granularity or not needs to be taken. They represent the value of embedded opportunity alone (e.g., to improve Filmflix performance). The numbers in the bottom cells of Figure 1 represent examples of *ROVs* calculated for real options embedded in Filmflix. On the other hand, the top parts of the cells represent

the overall architectural value including the real option values in the corresponding timestamp. The numbers in the top cells of Figure 1 represents examples of *AVs* calculated if a decision to pursue granularity or not is made focused on QoS improvements in Filmflix. These *AVs* are calculated using utility trees such as the one shown in Figure 2.

Figure 2 shows the likely improvement to the overall architecture's utility upon investing in a design decision (either adapting or not adapting granularity) at different points in time with the aim of delivering on architectural QoS improvement. For example, the utility tree numbers in Figure 2 quantify the improvement in architectural value of Filmflix if the decision to pursue granularity adaptation or not is made focused on QoS improvement.

The output of a binomial real options analysis tree is the value in the bottom part of cell A in Figure 1. ROV_0 represents the present real option value if it were to be exercised once at the final timestamp of a single binomial tree — F_t. This output is produced at the end of 2 stages in Figure 1: 1) the forward pass, which involves calculating the *AVs* starting from cell A then, 2) the backward pass, which involves calculating *ROVs* starting from cells B and C then moving backward. In the context of Filmflix, ROV_0 can mean the present value of improving the performance the architecture if granularity adaptation (or not) was pursued at timestamp F_t .

We interchangeably refer to the timestamp in which an option is exercised as the last timestamp of a binomial tree (F_t) or the option exercising timestamp (*et*). In the following paragraphs, we explain each component of the binomial trees in more detail. We use the values in Figures 2 and 1 as running examples to clarify each component.

Architectural value (AV): traditionally architectural value is likened to a financial stock's value. Architectural value quantifies the potential value of the overall architecture [48] when an improvement in an architectural utility (e.g., performance provision) would be obtained by granularity adaptation or retained by keeping the current level of granularity. Each *AV* quantifies the potential value of the overall architecture including utility in the corresponding utility tree cell. Values in the utility tree cells are elicited from architects' expectations, publicly available cross-company data, and/or architects' previous experiences with similar architectures. Each architectural value (AV_t) in each timestamp t is calculated using Eq. (1) in the first stage — the forward pass in Figure 1 — of constructing the tree.

$$AV_t = Value_0 + V_{Tree} \tag{1}$$

where $Value_0$ is the assumed value of the microservice architecture before we apply the options analysis. We acknowledge that $Value_0$ can represent architectural value whenever architects decide to utilise binomial real options analysis. Our assessment of it is consistent with the classical work of [48], where that value is used based on cost-benefit analysis of the satisfaction of the architecture to concerns of interests, where the valuation is elicited through the involvement of experts judgements and/or though the review of system operational logs (if available). This is certainly domain and case specific; it also depends on the scenarios used for valuation. For our work, we assume that the baseline value is provided, either systematically (e.g., through the use of widely used architecture evaluation methods for cost and benefits - e.g., ATAM [30]) or by using back-of-the-envelope expert judgement on the initial value potentials. Similar to the classical work of [48], we consider that expert judgements, publicly available data benchmarks, and/or previous experiences of microservice adopters are used to elicit $Value_0$.

V_{Tree} is the value V of the cell in utility tree *Tree* corresponding to binomial tree cell for which AV_t is calculated following traditional approaches in software engineering. In Figure 1, we consider the $Value_0$ to be 1750 by using AWS pricing tariffs for estimations. Consequently, the *AV* in cell A for example in the result of adding 1750 to 239.4. (the value in cell A of Figure 2)

Real option value rise rate (R_{ROV}) or fall rate (F_{ROV}): this refers to the expected rate of real option value rise or fall at timestamp *t* with respect to timestamp *t-1* [3, 48]. In the context of Filmflix for example, this refers to the rate at which

the value of scaling Filmflix — the example real option embedded in Filmflix — would rise or fall across timestamps. It corresponds to the improvement/deterioration in the architectural value at timestamp *t* with respect to timestamp *t-1*, calculated as follows:

$$R_{ROV} = AV_{raised_t} / AV_{t-1} \tag{2}$$

$$F_{ROV} = AV_{reduced_t} / AV_{t-1} \tag{3}$$

where AV_{raised_t} is the raised architectural value *AV* for current timestamp *t*. AV_{t-1} is the architectural value at the previous timestamp *t-1*. $AV_{reduced_t}$ is the reduced architectural value *AV* for current timestamp *t*. In our work, we use the same definition and formulas for real option value rise/fall rates as traditionally used in [3, 48]. R_{ROV} *and* F_{ROV} values in Figure 1 are calculated using Equations (2) and (3). For example, 1.03 is the result of dividing 2052.5 by 1989.4 as per (2).

Probability of real option value rise $P(Rise_{ROV})$ / $P(Fall_{ROV})$*:* this is a risk-neutral probability of real option value rise/fall, calculated based on Equations (4) and (5), respectively. The calculation includes a pre-defined risk-free interest rate *IR* representing the expected fluctuations in demand for the system [48]. The value of *IR* can be elicited from architects' expectations or publicly available cross-company data.

$$P(Rise_{ROV}) = \frac{(1 + IR)}{(R_{ROV} - F_{ROV}) * (100 - F_{ROV})} \tag{4}$$

$$P(Fall_{ROV}) = 1 - P(Rise_{ROV}) \tag{5}$$

The $P(Rise_{ROV})$ and $P(Fall_{ROV})$ values in Figure 1 are calculated using Equations (4) and (5). We consider the *IR* to be 0.01 in Figure 1. 0.51 in this Figure is the result of applying Eq. (4).

Real option value (ROV): the real option value quantifies the potential added value if the option is exercised. In the context of Filmflix this could be the value of exercising the option to scale Filmflix. The real option values in a single binomial analysis tree are calculated using the equations below in the second stage — backward pass in Figure 1 — of constructing the tree. The real option value (ROV_{et}) in the final timestamp/option exercising timestamp *et* of a tree is calculated using Eq. (6). All ROV_t where $t \neq et$ are calculated using Eq. (7).

$$ROV_{et} = MAX(0, AV_{F_t} - MC) \tag{6}$$

$$ROV_t = \frac{ROV_{raised_{t+1}} * P(Rise_{ROV_t})}{100} + \frac{ROV_{reduced_{t+1}} * P(Fall_{ROV_t})}{100} * \frac{(1 + IR)}{100} \tag{7}$$

where AV_{F_t} is the architectural value at F_t and *MC* is the estimated maintenance cost of the architecture. $ROV_{raised_{t+1}}$ is the raised and $ROV_{reduced_{t+1}}$ is the reduced real option value at the following option exercising timestamp *t+1* of the backward pass. In Figure 1, *ROVs* are calculated using *AVs* in the corresponding cells. We consider the *MC* for these calculations to be 500.

We assume the maintenance cost to include the activities required for the health of a running architecture[12, 48] (i.e., the adapted or un-adapted one) rather than those required for adapting its granularity. These include development, configuration, securing communication links, and implementing data translation layers. Our process is flexible, making it possible to integrate coarse-grained cost models (e.g., CostHat [37]) or finer-grained ones, if historical data is available (e.g., expert judgements, cross-company data, what-if analysis using a simulated ambient environment) to calculate the maintenance cost. The maintenance cost estimation is assumed to be constant, unless further data becomes available. At the end of the backward pass, the bottom part of the base cell in the tree (cell A in Figure 1) shows the present

real option value if the option were to be exercised only at the last timestamp F_t. However, as the potential value of exercising an option is time sensitive, we have assumed the American view of options (i.e., option can be exercised at any point before it expires [3]). Exercising an option can enable multiple options (i.e., compound options[29]). Total real options ROV_{total} is calculated using Eq. (8):

$$ROV_{total} = (\sum_{i=0}^{i=F_t} ROV_{t=0_i}) - RC \tag{8}$$

where i is a counter for the intermediate binomial trees, ranging from 0 to the final timestamp F_t and RC is the re-structuring cost — the cost of pursuing granularity adaptation. This cost may include taking the microservices of concern offline, then encoding and enforcing a chosen granularity adaptation strategy and finally re-deploying the adapted architecture. Similar to maintenance cost, several fine- or coarse-grained cost models can be used to elicit the re-structuring cost for all of these activities related to pursuing granularity adaptation. Eq. (8) adds the real option value in the base cells ($ROV_{t=0_i}$) of multiple binomial trees (each capturing a different range of timestamps) then deducts the re-structuring cost. Our process is flexible, allowing using cost model to estimate the re-structuring cost (e.g., expert judgement and/or cross-company-data) when calculating the re-structuring cost. The upper bound of the maximum range covered by intermediate trees (i.e., F_t in Eq. 8) corresponds to the real option's expiry date, after which the real option (e.g., scaling the architecture by replication across rented application servers) becomes unavailable to exercise (due to the application server's lease end for example). $ROV_{t=0_i}$ is the real option value in the base cell t=0 of the i^{th} intermediate binomial tree.

To take a decision of adapting the granularity or not, the architect can compare ROV_{total} of the real options in the adapted and un-adapted architectures at runtime. If adapting has larger ROV_{total}, it is worth adapting the granularity. Section 4.4 shows how to combine real options analysis with Bayesian surprises, so that ROV_{total} can be adapted at real time in response to runtime evidence.

4.3 Utilising Bayesian Surprises

Fluctuations in QoS provision at runtime tend to affect the utility of the system which can impact added value. For example, fluctuations in Filmflix's performance can impact end users' rate of uploading reviews, thereby impacting the added value embedded in Filmflix's architecture. Therefore, estimates of rise/fall probabilities in real option values based on expert judgements or cross-company data for example can suffer from under- or over-estimation due to the dynamic behaviour of architecture and/or the architects' imperfect knowledge of the highly dynamic microservice environment.

Therefore, instead of taking a fixed estimation for the probability of rise/fall in real option values, we continuously update these probabilities at runtime making use of runtime observations of QoS provision and fluctuations of workload. For example, consider a situation where runtime observations of Filmflix invocation duration and incoming movie reviews show that a high volume of reviews can negatively impact Filmflix's invocation duration. This in turn can raise the probability of real option value rise associated with pursuing granularity adaptation aiming to improve Filmflix invocation duration. This rise is effective provided that a high volume of requests is expected to continue.

Bayesian surprises quantify the divergence between beliefs (claims [9]) about the system's behaviour prior and posterior to observing runtime evidence [9]. No divergence means the belief is verified leading to zero Bayesian surprise. The larger the magnitude of the surprise the more evidence there is regarding the belief violation. The concept of Bayesian surprises has been applied in the context of self-adaptive systems (e.g.,[49] and [25]). In these

applications, Bayesian surprises are used to "identify new situations (regarding the software architecture's behaviour or the environment in which it operates) where the current preferences may need to be re-evaluated to improve the levels of satisfaction of non-functional requirements [49]". Bayesian surprises have not been applied in the context of microservices nor have they been combined with real option analysis previously. We use Bayesian surprises to quantify the significance of runtime evidence in updating beliefs about the system's likely runtime behaviour. We then use Bayesian surprises to update the probabilities of rise/fall in real option values. Our work is the first to combine Bayesian surprises with real options analysis and the first to apply it in the context of microservice granularity adaptation.

In this work, we consider a claim that articulates uncertain beliefs in the context of end user workloads. Assume the scenario where Netflix receives around one billion streaming requests everyday [27]. In this context, the end users are Netflix subscribers, the workload is the amount of streaming requests and the claim articulates the architects' belief about how much the QoS provision of the overall architecture is affected by fluctuations in the streaming workload. A claim capturing this can be articulated as follows:

- if end user workloads are greater than a pre-defined workload threshold then the QoS provision always becomes worse than a QoS threshold and,
- if QoS provision is worse than a QoS threshold, this is always caused by the end user workload exceeding the pre-defined workload threshold.

Other claims can be articulated assessing the impact of other contexts (e.g., organisational changes) on QoS.

When the binomial trees are initially constructed in our process, they are constructed assuming a claim holds true. However, this claim can be assessed at runtime; if it is violated this calls for updating the trees at runtime since behaviour of the architecture is not as initially expected. Thereby the values which can be enabled by adapting granularity or not (which are captured in the binomial trees) are not as initially believed. We acknowledge that this route could be supplemented in future research with updating the claim thresholds to reflect the runtime evidence on the architects' beliefs. Below, we explain several concepts necessary to understand how Bayesian surprises work.

Runtime evidence variables (REV): these are variables monitored at runtime; their values indicate violation or verification of claims. Runtime evidence variables are pairs of each end user workload and corresponding QoS attribute value recorded at runtime. These pairs are chosen to observe whether the architecture meets the pre-defined thresholds articulated in the claim or not at runtime.

Claim prior probability P(Claim): this is the probability of a claim to hold true before observing runtime evidence variables [9]. For example, there is no means of ensuring at design time whether QoS provision will indeed be within the pre-defined threshold and the workload will be as articulated in the claim. Therefore before a microservice architecture is deployed, the claim prior probability holding true is assumed to be 0.5. Since there is no runtime evidence observed, there is an equal likelihood of the claim holding true or being violated at runtime. Thereafter, the prior probability of the claim is carried over at runtime before observing new runtime evidence variable values. At a high level therefore the *P(Claim)* value depends on the availability of evidence related to the architecture's behaviour prior to assessing the claim.

Claim posterior probability P(Claim | REV): this is the probability of the claim to hold true assessed in the light of observing relevant evidence *(REV)*. In our process, the claim posterior probability is only different from the prior probability if the runtime evidence violates the claim. Our process is flexible regarding the method used to estimate the posterior probability. Crucially though, this method has to capture the dependence between runtime end user workload and QoS recorded values (e.g., by using Bayesian networks [55]) rather than assuming independence between them

(e.g., Naive Bayes [55]). In context of the example claim above, QoS provision depends on the workload and the claim holding true in turn depends on both.

Bayesian surprise (S_{claim}): this is the divergence between the claim prior and posterior probabilities, calculated using Eq. 9 [9]:

$$S_{claim} = P(Claim|REV) * \log \frac{P(Claim|REV)}{P(Claim)} \tag{9}$$

Bayesian surprises only measure the magnitude of the divergence given runtime evidence variable values (*REV* in Eq. 9) regardless of the direction (good or nasty surprise). The larger the magnitude of S_{claim}, the larger the update regarding beliefs (articulated in the form of claims) about the system behaviour. S_{claim} can take continuous values starting from zero.

In this paper, we elicit this direction by comparing the trends of QoS attribute values against end user workload values. A good surprise occurs if the QoS attribute values improve over time even though the end user workload is above the pre-defined workload threshold. A good surprise promotes keeping the current microservice granularity level. For example, if the QoS provision is within the QoS threshold articulated in the claim when the workload is beyond the respective threshold in the same claim, this is a good surprise. In other words, the microservice(s) is behaving better than expected, calling for not adapting. Therefore if S_{claim} is good, it is used to update the binomial tree of not adapting.

A nasty surprise occurs if the QoS attribute values deteriorate over time even though the end user workload is below the pre-defined workload threshold. Intuitively, a nasty surprise promotes granularity adaptation. For example, if the QoS provision is beyond the QoS threshold articulated in the claim when the workload is within the respective threshold of the same claim, this is a nasty surprise. In other words, the microservice(s) is behaving worse than expected, calling for granularity adaptation. Therefore if S_{claim} is nasty, it is used to update the binomial tree of adapting.

Bayesian surprise tolerance threshold — in percentage (T_S), in corresponding exact value (T_s^{exact}): in[25], a surprise tolerance threshold is pre-defined for every claim to indicate the criticality of the QoS attributes captured in it to the software architects. Only if the Bayesian surprise triggered on the claim exceeds this threshold will action be taken. In our context, this action is updating the probability of option rise for granularity adaptation or for keeping the current architecture at runtime. For every claim, this tolerance threshold is defined at design-time as a percentage by the software architect (T_S in Eq. 10). The higher this percentage the less critical the QoS attribute is to the software architects. The exact value (T_s^{exact} in Eq. 10) to which this percentage corresponds is calculated during every runtime iteration of our process given a record of the Bayesian surprises triggered in the previous runtime iterations using Eq. 10.

$$T_S^{exact} = T_S * (S_{claim_{max}} - S_{claim_{min}}) \tag{10}$$

where $S_{claim_{max}}$ is the maximum triggered Bayesian surprise on the claim so far and $S_{claim_{min}}$ is the minimum triggered Bayesian surprise on the claim so far. $T_{S_{exact}}$ always gets tighter as more runtime evidence is acquired. The larger the range of triggered surprises, the larger the difference between one T_s^{exact} and the next.

4.4 Proposed Dynamic Evaluation Process

Our novel process determines whether or not it is worth pursuing adaptation at each runtime iteration of our process. We contribute to a novel evaluation process for dynamically evaluating the worth of pursuing granularity adaptation. It uses the runtime evidence to update at real time the potential added value of pursuing granularity adaptation. It extends traditional real options analysis, thereby being the first contribution which enables using real options analysis at runtime. Bayesian surprises are used to update the probabilities of real option rise or fall in traditional binomial trees.

This enriches traditional binomial real option analysis with the impact of runtime evidence on the real option values. In our process, the added value calculation continuously calibrates evidence variables observed over runtime iterations.

In this section we explain the inputs, process, and outputs of our process. We complement this explanation with Appendix A which includes a more concrete example of applying our process. It uses the same setting which we use in our evaluation. Appendix A shows which values are plugged into which equations in order, and how the equations lead our process's output in a single iteration.

4.4.1 Inputs. We formulate the inputs of our process benefiting from the concepts described in Sections 4.2 and 4.3. Our inputs capture the potential likely cost and benefit as well as runtime dynamics related to adapting and not adapting granularity. Software architects can follow these steps when providing input to our process.

(1) Within the microservice monitoring tool of interest, software architects need to determine which microservices are going to be monitored at runtime using this tool and reasoned about using our process. Our process is flexible regarding the scope of microservices input to it; the input could be a single microservice or multiple interdependent microservices. It is worth noting that all inputs in the following steps will be concerning the scope determined in this step. For example, maintenance and re-structuring costs are estimated for the overall pre-defined scope of concern. The same reasoning applies to the inputs related to calculating Bayesian surprises. The pre-defined scope is monitored at runtime for QoSs of concern. Where the scope entails multiple microservices the overall QoS is monitored at runtime rather than the individual microservices.
(2) Within a tool support for our process, the software architect needs to provide the following inputs for claim articulation (see Section 4.3):
    (a) *Runtime evidence variables*: the QoS attribute indicated by the architects to be monitored at runtime to assess the validity of the claim. Without loss of generality, we assume that larger values indicate worse QoS. We consider the end user workload to be the other runtime evidence variable to be recorded at runtime along with the indicated QoS attribute. End user workload will measure the runtime end user interest in the QoS attribute indicated by the software architect;
    (b) *QoS attribute threshold:* the QoS value above/below which QoS of the monitored microservice(s) is considered by software architects to be unacceptable. A single QoS threshold is required for each claim to be assessed by our evaluation process. The exact threshold value depends on criticality of the QoS attribute to the software architect; the more critical a QoS attribute, the tighter the value of QoS attribute threshold would be.
    (c) *End user workload threshold:* the end user workload value below/above software architects expect the monitored microservice(s) to miss a QoS attribute threshold. A single end user workload threshold is needed for each articulated claim. The exact end user workload threshold for each claim can be determined by software architects' experience regarding the microservice environment, expert judgements, publicly available data benchmarks, and/or previous experiences of microservice adopters.
(3) Within a tool support for our process, the software architect needs to provide inputs for Bayesian surprise calculation (see Section 4.3):
    (a) *Claim prior probability,* and, b) Bayesian surprise tolerance threshold.
(4) Within a tool support for our process, the software architect needs to provide inputs for binomial tree construction (see Section 4.2):
    (a) *Re-structuring cost*, b)Maintenance costs for the adapted and un-adapted architectures, c) Initial architectural value of the adapted and un-adapted architecture, d) Utility trees for adapting and not adapting, and, e) Option

expiry date: the predicted number of binomial tree timestamps (in unit days, months or years) corresponding to the option expiry date. The same numeric value is used for the un-adapted/adapted architectures for fair comparison of the total real option values.

In Section 5.4, we report on a tool support which can be used to follow the above.

4.4.2 Process. We contribute to a novel evaluation process for dynamically evaluating the worth of pursuing granularity adaptation. In every runtime iteration of our process, it solicits and monitors runtime evidence then feeds it into calculating and updating added value. Pseudocode (shown in Alg. 1) — AddedValueUpdate — presents our proposed evaluation process and Table 1 defines the acronyms used in the pseudocode. Initially, AddedValueUpdate assumes

Algorithm 1 The pseudocode presenting our proposed process; Table 1 defines the acronyms used in the pseudocode

1: **procedure** AddedValueUpdate
2: **Input:** *RC, MCs Value$_0$,Trees,F_t, $Q_{ri\ name}$, T_w, T_q, P_{claim}, T_s*
3: Claim: [$W_{ri} > T_w \leftrightarrow Q_{ri} > T_q$]
4: **for** each ri **do**
5: **if** ri=0 **then**
6: Construct Binomial Real Options Trees for Adapting and Not Adapting using $Value_0$, Trees, F_t, MCs
7: **else**
8: Carry Binomial Real Options Trees for Adapting and Not Adapting from ri-1
9: **end if**
10: Solicit and Monitor Q and W
11: Calculate S_{claim} and $T_{s\ exact}$
12: **if** S_{claim}=0 OR ($S_{claim} < T_{s\ exact}$) **then**
13: Go to next iteration
14: **else if** ($Q_{ri} > Q_{ri\text{-}1}$) AND $W < T_w$) **then** ▷ *nasty surprise*
15: $P(Rise_{ROV}{}^{adapting}) * S_{claim}/T_s^{exact}$
16: **else if** ($Q_{ri} < Q_{ri\text{-}1}$) AND $W > T_w$) **then** ▷ *good surprise*
17: $P(Rise_{ROV}{}^{not\ adapting}) * S_{claim}/T_s^{exact}$
18: **end if**
19: Re-calculate $ROV_{total}{}^{adapting}$ and $ROV_{total}{}^{not\ adapting}$
20: **if** $ROV_{total}{}^{adapting} > ROV_{total}{}^{not\ adapting}$ **then**
21: $Suggestion_{ri}$=Yes
22: **else**
23: $Suggestion_{ri}$=No
24: **end if**
25: **Output:** $ROV_{total}^{adapting}$, $ROV_{total}^{notadapting}$, $Suggestion_{ri}$
26: **end for**
27: **end procedure**

the software architects formulate their belief about the system's runtime behaviour in the form of a claim articulated as Line 3. A claim is articulated using an input end user workload threshold (T_q) and a corresponding input QoS attribute threshold (T_q). For example, consider the following claim: "workload > 100 requests ↔ Invocation duration > 1 millisecond". This claim is read in two ways:

- if a microservice(s) receives more than 100 parallel requests, then the invocation duration will always be more than 1 millisecond and,

Table 1. AddedValueUpdate acronym definitions

Acronym	Term
RC	Re-structuring cost of granularity adaptation
MCs	Maintenance costs for the adapted and un-adapted architectures
$Value_0$	Initial architectural value
Trees	Utility trees for adapting and not adapting
F_t	Final timestamp of a single tree (or overall option expiry date)
$Q_{ri\ name}$	QoS attribute name at runtime iteration ri
T_w	Workload threshold
T_q	QoS attribute threshold
P_{claim}	Claim prior probability
T_s	Bayesian surprise tolerance threshold
t	Binomial tree timestamp counter
ri	Runtime iteration counter
Q_{ri}	The value of $Q_{ri\ name}$
W_{ri}	End user workload value at runtime iteration ri
$ROV_{total}^{adapting}$	The total real option value for adapting
$ROV_{total}^{not\ adapting}$	The total real option value for not adapting
$Suggestion_{ri}$	Final verdict of the pseudocode for pursuing granularity adaptation in ri

- if the invocation duration of the microservice(s) is more than 1 millisecond, this is always due to the number of parallel requests exceeding 100.

The claim is assessed and subsequent analysis is carried out in each runtime iteration (Line 4). The length of time between runtime iterations determines the rate of soliciting runtime evidence, the rate of updating added values, when adaptation is pursued. This length depends on the criticality of the application and/or runtime scenario. For example, when Filmflix is operating during holiday seasons under high workloads then even a short time with poor QoS could lead to serious consequences.

Therefore, the rate at which this QoS needs to be monitored is high and thereby the length of time between runtime iterations needs to be in the order of milliseconds. It is worth noting that the length between runtime iterations is different from the option exercising timestamp. The option exercising timestamp is a point in time (not duration) when the option is exercised. For example, consider a real option which enable adding a new functionality in Filmflix. The point in time when this new functionality is added (i.e., the real option is exercised) is the option exercising timestamp.

In each runtime iteration, binomial real options analysis trees are first constructed to calculate the total potential real option value which can be enabled adapting and retained by not adapting granularity using the inputs and equations explained in Section 4.2. For the first iteration, the calculation is done using traditional binomial real options analysis (Line 6). For each following iterations, no manual construction is required. Instead, the most-up-to-date binomial trees are carried over from the end of previous iteration (Line 8). Then the quality Q_{ri} and workload W_{ri} values are recorded (Line 10) and analysed to update probabilities of option rise in the binomial trees if need be (Lines 14 to 18).

The analysis is done by calculating Bayesian surprise S_{claim} regarding the claim and the exact Bayesian surprise tolerance threshold T_s^{exact} (Line 11). No probability update is required if the Bayesian surprise is zero (the claim is verified) or below T_s^{exact} (Line 12). Otherwise, option value rise probability updates will be performed. Using the same claim example above, if the invocation duration is 1.05 milliseconds when there are 101 parallel requests, the claim is violated. Therefore, S_{claim} is greater than zero in this iteration. However, if T_s^{exact} is greater than S_{claim}, then this violation of the claim is not significant enough to have an impact on the adaptation decision.

Intuitively, a nasty surprise promotes granularity adaptation. For example, if the invocation duration is 2.5 milliseconds when there are only 75 parallel requests, this is a nasty surprise. In other words, the microservice(s) is behaving worse

than expected, calling for granularity adaptation. Therefore, the probability of option rise in adapting granularity is increased if a nasty surprise is encountered (Line 15). The factor by which the probability is increased depends on the significance of the Bayesian surprise. A more significant Bayesian surprise is further away from the surprise tolerance threshold than a less significant surprise. For example, if the invocation duration is 10 milliseconds for 50 parallel requests, this is a more major claim violation than if the invocation duration is 1.75 milliseconds for 95 parallel requests. We reflect this significance as a fraction of the calculated Bayesian surprise over the exact surprise tolerance threshold (S_{claim} / T_s^{exact}).

Conversely, good surprise promotes keeping the current microservice granularity level. For example, if the invocation duration is 0.8 milliseconds when there are 110 requests, this is a good surprise. In other words, the microservice(s) is behaving better than expected, calling for not adapting. Therefore, the probability of option rise in not adapting granularity is increased by the (S_{claim} / T_s^{exact}) factor if a good surprise is encountered. Once the probabilities of option rise and fall are updated according to the direction of the Bayesian surprise, the updated total real option values are calculated which correspond to adapting and not adapting (Line 19).

4.4.3 Outputs. At the end of each runtime iteration, the pseudocode suggests adapting granularity (Line 21) or keeping the current architecture (Line 23). The output is presented as a final verdict regarding pursuing granularity adaptation ($Suggestion_{ri}$) and the total real option values used to arrive at this verdict (Line 25). Following the process's suggestion in this case is a cautious approach to avoid unjustified adaptation that will not introduce added value to the architecture but rather incur unnecessary re-structuring and maintenance costs. In this case, all the inputs to the current runtime iteration are carried over to the next one.

On the other hand, if the total real option values corresponding to adapting are greater than not adapting (Line 20), the pseudocode outputs a suggestion to pursue adaptation in runtime iteration *ri* (Line 21) along with the total real option values used to arrive at this verdict (Line 25). Following the process's suggestion to pursue adaptation in this case is a cautious approach to introduce the added value in the real options rather than taking a reactive route which could result in missing opportunities for economic gains.

The output real option values represent the potential added value of pursuing (or refraining from) granularity adaptation (thereby answering RQ1). Our process captures uncertainty regarding this potential added value — due to workload fluctuations or unexpected microservice behaviour for example — by utilising binomial real options analysis. By definition, binomial trees incorporate the probability of real option values rising and falling over time (thereby answering RQ2). Our process updates uncertainty regarding potential added value by carrying out Lines 12-19 of Alg. 1 (thereby answering RQ3).

5 EVALUATION

We evaluate the extent to which our contribution addresses the problem formulated in Section 3. We leverage goal-question-metric (GQM) approach [11] to provide systematic guidance for identifying the metrics that can drive the evaluation relative to goals of interest:

(1) Goal: The goals of our evaluation are to reveal (1) the unique benefit of our evaluation process over state-of-the-practice in reasoning about microservice granularity and, (2) the expressiveness of our evaluation process compared to traditional binomial analysis in terms of capturing runtime dynamism of microservice architectures and reflecting that dynamism on added value calculations.
(2) Question: Related to the first goal above the questions of concern are: *Q1: What inputs are required and what outputs are available in our evaluation process but not in other tools?* and *Q2: Why are the unique inputs and/or outputs of our process beneficial for reasoning about microservice granularity?* Related to the second goal above the question of concern is: *Q3: How does runtime evidence change the output of our process and that of traditional binomial analysis?* Sub-questions include:
    - When is a surprise triggered in our approach?
    - Does triggering surprises lead to a more informed verdict than that of traditional binomial analysis?
(3) Metric: To answer Q1, we have defined the following metrics: M1: The number of inputs of our process. M2: The number of inputs of the industrial tool of concern. M3: The number of outputs of our process. M4: The number of outputs of other tools compared to our process. In other words, we measure whether there are more inputs and/or outputs in our process compared to other tools to highlight which inputs/outputs are unique to our evaluation process. To answer Q2 above, these metrics can be used to compare and to discuss the implications of these input/output differences thereby highlighting why these unique features are beneficial for addressing the target problem of this paper. Related to Q3 above the metrics of concern are: M5: trends in ROV in response to trends in runtime evidence and, M6: point-wise changes in ROV compared to traditional binomial analysis. M5 shows how runtime evidence changes the output of our process (addressing the first part of Q3). M6 shows how runtime evidence changes the output of traditional binomial analysis (addressing the second part of Q3).

We facilitate investigating the metrics by setting up a controlled experiment on a Filmflix case study. We consider controlled experiment on Filmflix to be the most appropriate evaluation means for investigating metrics of concern; it provides significant scale for answering them. In particular, looking at microservice-related literature (e.g., [1, 19, 22, 24, 31, 32, 42, 43, 58], and [15]), we observed that microservice-specific proposed architectural design support processes can be evaluated on case studies of comparable scale and scope of functionality to Filmflix.

We implement the initial Filmflix architecture using AWS Lambda. AWS Lambda "is a computing service that lets you run code (which fits different templates — including a microservice template) without provisioning or managing servers [7]." AWS Lambda wraps the source code as a Lambda function. We inject three different trends of end user workload (stepwise, single spike and multiple spikes) into the Lambda function at runtime. This variation in trends allows us to show clearly how the process reflects runtime dynamics on the calculation of added value of granularity adaptation. We thereafter record the QoS attribute values using AWS CloudWatch [6] for each workload trend and use the records for our evaluation.

In Section 5.1 we describe a controlled experiment setting. Using the same setting, we apply our process and traditional binomial analysis separately when injected with each workload trend described above.

In Section 5.2 we compare our process's output against the output of AWS Cloudwatch — a representative state-of-the-art microservice runtime monitoring tool — in the controlled experiment setting. The comparison shows tangible evidence of our process's benefits in linking the technical decision of granularity adaptation to its potential added value (thereby addressing the first goal above). Additionally, we discuss the features provided by three other state-of-the-practice tools — Rieman, X-Pack, and IBM Bluemix — versus the features provided by our process.

In Section 5.3, we use the setting from Section 5.1 to compare our evaluation process with traditional binomial real options analysis. Our comparison shows that our process goes beyond traditional real options analysis by successfully performing real time updates of added value in response to runtime evidence variable values (e.g., workload variations). Successfully reflecting runtime dynamics means our process can track and update at runtime the potential added value of microservice granularity adaptation.

In Section 5.4, we report on a Java implementation of our process to show its feasibility as a tool support after having discussed how our contribution meets the evaluation goals.

5.1 Experimental Setup

We consider the following setting of parameters for a controlled experiment that applies our process's calculations on Filmflix's initial architecture. The parameters described below map partially to the inputs listed in Section 4.4.1. In particular, there are some additional points here to give context to the way we set up the experiment (e.g., runtime iteration duration, monitored microservices). Since we are applying our process in a controlled experiment setting, we consider tighter than typical values for the claim thresholds on invocation duration and review string lengths provided to our process as input. These tight values give an insight into how our process operates on both significant and non-significant Bayesian surprises given different workload trends. These insights can also be transferable to cases where the claim thresholds are typical rather than tight.

- *Microservice:* Since MovieReview encapsulates other microservices in the initial architecture, we choose to monitor MovieReview at runtime to get a comprehensive view of the system's behaviour given different workload trends. However, we do acknowledge that monitoring larger or smaller parts of the architecture can serve other rationales (e.g., pinpointing the cause of a malfunctioning system).
- *QoS attribute (a runtime evidence variable):* We assume microservitization is driven by enhancing high performance provision. We measure performance using the invocation duration of MovieReview, monitored at every runtime iteration of our evaluation process.
- *Workload (a runtime evidence variable):* Each review string corresponds to a separate call to the MovieReview AWS Lambda function. Since MovieReview encapsulates two other microservices, each review string corresponds to two internal invocations to ReviewUpload and ReviewRegulation. To show how our process reflects the workload fluctuations on the calculation of added value at runtime, we consider workload as the written review string submitted by the end user. Furthermore, we control this workload at runtime by varying each review string length. We separately inject three different trends of review string lengths (step-wise increase in lengths, single spike, multiple spikes with complying and non-complying review strings). Step-wise increase means every couple of injections the length of the string is increased by a fixed number of sentences. Single spike workload means that most string lengths vary by a fixed small number of sentences. Only one string in this workload is much longer than the others creating a spike. Multiple spikes follows a similar trend to the single spike except there are two injections where the injected string is much longer than the rest of the workload. In the real world, the workload will be monitored rather than

controlled. However, controlling one of the runtime evidence variables by injecting different trends of end user workload allows evaluating the effectiveness of our process in capturing the primitive cases of runtime dynamics and reflecting them on the added value calculations. In an industrial setting, we envision that workload fluctuations would be permutations of these cases.

- *QoS attribute threshold:* To ensure that significant Bayesian surprises are triggered at runtime, we assume a tight invocation duration threshold of 1 millisecond when articulating the claim to be assessed at runtime.
- *End user workload threshold:* To ensure that a range of significant and un-significant Bayesian surprises are triggered at runtime, we use 100 sentences as the workload threshold when articulating the claim. We infer this threshold by testing our implementation of the initial architecture. The testing shows that for reviews more than 100 sentences long, MovieReview calls for extra regulatory reviews from ReviewRegulation.
- *Claim:* Given the thresholds above, we consider the claim to be assessed at runtime as a claim using if and only if ($\leftrightarrow$): "review string length > 100 sentences $\leftrightarrow$ Invocation Duration > 1 millisecond". We acknowledge that these claims can be set in the future as alarms within in AWS CloudWatch to alert our evaluation process whenever a claim is violated. In that case, our evaluation process would become more event-based rather than iterative.
- *Bayesian surprise tolerance threshold:* To stress test our process, we assume that improving performance is critical to the software architects, calling for a tight Bayesian surprise tolerance threshold (15%).
- *Claim posterior probability:* We use Bayesian networks [55] to calculate the claim posterior in each iteration. Bayesian networks allow us to capture the dependency between review string length, invocation duration, and the truth of the claim. The posterior probability of the claim holding true (*P(Claim | REV)* in Eq. (5.1)) is derived from frequency tables. Each record in the table has a combination of invocation duration, review string length (the *REV*) ranges and the corresponding number of times the claim held true ($t_{claim}^{StringLength,InvocationDuration}$), and/or was violated ($f_{claim}^{StringLength,InvocationDuration}$) within that range. The posterior probability is derived from the correct record of the frequency table as:

$$P(Claim|REV) = \frac{t_{claim}^{StringLength,InvocationDuration}}{t_{claim}^{StringLength,InvocationDuration} + f_{claim}^{StringLength,InvocationDuration}} \quad (11)$$

- *Runtime iteration duration:* Assuming that improving the performance is critical to software architects and given that the QoS attribute threshold is in the order of milliseconds, the length of each runtime iteration of our process needs to be in the order of milliseconds as well. Therefore, we assume the runtime iteration is 5 milliseconds long. We inject a single review string in each runtime iteration.
- *Option expiry date:* The option expiry date dictates the number and depth of constructed binomial trees. Therefore, we consider three timestamps as the option expiry date to avoid prohibitive computational costs in each iteration of our process. In other words, an option enabled by adapting or retained by not adapting will be available for three runtime iterations after pursuing adaptation.
- *Maintenance cost of the adapted architecture:* we use CostHat [37] (explained in Section 4.2) to estimate the maintenance cost if granularity is adapted. We define the service call graph for a sample adapted architecture — where MovieReview and ReviewRegulation are merged into a single microservice — and calculate the maintenance cost for this sample. In particular, there would be calls cascaded from MovieReview to and from ReviewUpload for each request from the end user to the application. Cascading calls in each direction cost an average of £150 when all four parameters of CostHat are considered. Therefore, the overall maintenance cost of the adapted architecture would be £300. We

acknowledge that this value can be further elaborated if the specific manner of adapting granularity is known and/or other more fine-grained cost models are used.

- *Maintenance cost of the un-adapted architecture:* we use CostHat to estimate the maintenance cost of keeping the initial architecture. For our case, this is the maintenance cost given that the MovieReview microservice cascades calls to both the ReviewUpload and ReviewRegulation microservices in the initial architecture and then ReviewUpload cascades a call back to MovieReview. Each cascaded call costs on average £170 using CostHat, totalling to an average of £500. We acknowledge other cost models can be used to estimate this cost.
- *Re-structuring cost:* we assume architect judgement is used to estimate re-structuring cost in our evaluation. There are three microservices in the initial architecture: ReviewUpload, ReviewRegulations, and MovieReview. Amending each microservice can require an average £500, totalling to £1500 for re-structuring the initial architecture. This cost can alternatively be elicited more accurately from publicly available cross-company data. However, this experiment is more concerned about illustrating the impact of runtime dynamics in our evaluation rather than soundness of the underlying cost estimations.

This setting covers situations where we expect to see nasty surprises, good surprises, and the claim being assessed.

5.2 Results: Capturing Added Value Under Uncertainty (Addressing First Evaluation Goal)

We highlight the unique benefits of our evaluation process thereby addressing the first goal of concern. Figure 3 shows the invocation durations of MovieReview when monitored using AWS CloudWatch for 14 runtime iterations in the case of stepwise workload increase. In Figure 3, every two increments on the x-axis corresponds to increasing the length of the review string by 30 sentences. After 20:00:00.25, the invocation duration decreases despite continuing to increase the workload stepwise. It is worth noting that the output of AWS CloudWatch is produced in milliseconds rather than in monetary terms (as is the case in our process). AWS Cloudwatch does not show the ramification on added value when answering the question of when to adapt microservice granularity (or not) at runtime.

Figure 4 presents the output of utilising our process; it shows the total potential real option values which can be enabled by adapting and by not adapting the granularity in the same runtime iterations given the invocation durations in Figure 3. Figure 4 provides a value-driven basis to reasoning about granularity adaptation decisions. For example, runtime iteration 5 shows the total potential real option value which would be enabled by adapting and the value retained by not adapting. As the option value is higher when adapting than when it is retained, therefore adapting is suggested at the end of iteration 5. The ability to take added value into account is a unique benefit of our approach.

Comparing Figures 3 and 4, only the latter provides value-driven justification for pursuing adaptation by presenting its ramifications related to added value. If Figure 3 is used alone to answer the question of whether to pursue adaptation or not in iteration 13, intuitively the architects would decide to pursue granularity adaptation to reduce the invocation duration which had increased over the previous two iterations. On the other hand, Figure 4 suggests keeping the current granularity level in iteration 13 despite the increase in invocation duration. Therefore, relying on AWS Cloudwatch alone gives less informative insight compared to our contribution's insights regarding the ramifications of granularity adaptation decisions.

In Figure 3 there is no spike in invocation duration at 20:00:00.40. However, the corresponding iteration 8 in Figure 4 promotes adapting since the maintenance cost of the un-adapted architecture is higher in this runtime iteration. If only AWS Cloudwatch is used to answer the question of whether to adapt or not in iteration 8, the intuitive answer would be not to adapt since there is no apparent technical benefit. The technical decision ignores the potential option value,

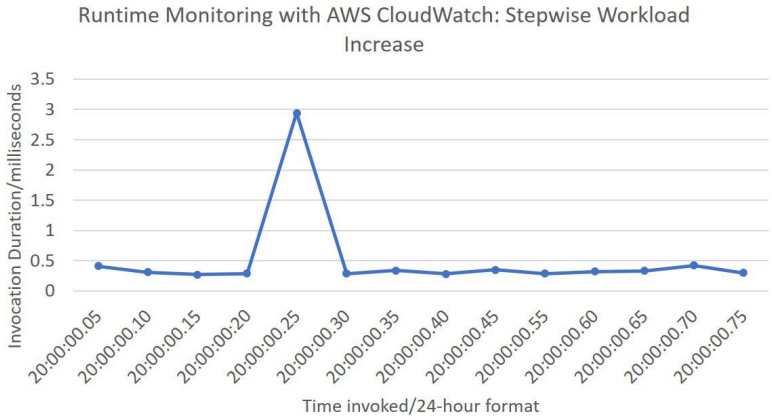


Fig. 3. AWS CloudWatch output for stepwise workload increase (i.e., increasing the review string by 30 sentences) injected to Filmflix in Section 5.1

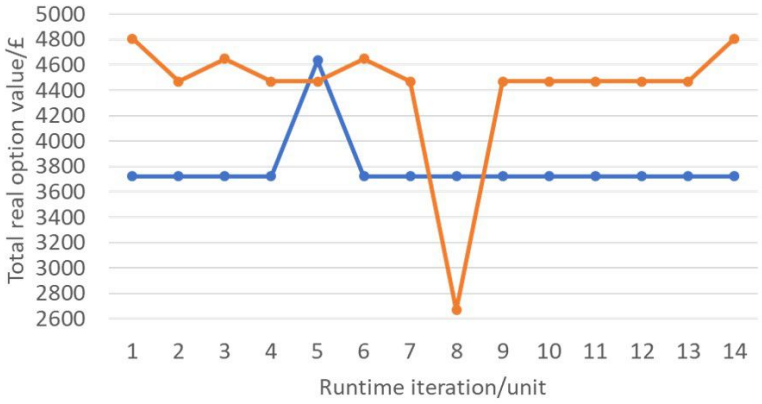


Fig. 4. Corresponding output of our process over same time period as in Figure 3; 1 runtime iteration = 5 milliseconds; the blue line refers to the real option values of adapting granularity while the orange line refers to the real option values of keeping the granularity level unchanged

that if factored in, would change the adaptation decision. Factoring cost factors in value-driven assessment means our contribution provides a more effective means to reason about granularity adaptation leading to a different decision altogether. Therefore, the output of our process is more informed compared to that of AWS Cloudwatch when used to reason about pursuing granularity adaptation. On the other hand, we acknowledge that the input to our process is more complex than that to AWS Cloudwatch. Looking at the metrics of concern, using our process creates a trade-off between complex input and beneficial output. On the other hand, using AWS Cloudwatch does not require complex input but does not provide as beneficial output as our process.

Further delving into Figure 4, we observe two iterations where adaptation was suggested: 5 and 8. Depending on the specific granularity adaptation strategy pursued, these consecutive suggestions can represent situations where a

strategy and its reverse are conducted. For example, consider a case where two microservices are merged in iteration 5 then the same ones are decomposed back as they were in iteration 8. Such cases can raise a point against merging these two microservices in the first place. However, had they have been left un-merged for iterations 6 and 7, these microservices could have led to potential reductions in added value for the overall architecture. Such reduction, for example, could be of high significance in safety-critical microservice architectures. Therefore, our process strives for introducing added value to the microservice architecture whenever possible even if for a small number of runtime iterations. It is worth noting that the length of each runtime iteration also has a high impact on the effectiveness of our process in suggesting adaptation at appropriate runtime iterations.

Looking at other industrial runtime monitoring tools, the X-Pack plug-in provides the most flexibility for monitoring the health of microservice applications at runtime [40] to our knowledge. In particular, this plug-in is compatible with cloud container providers (e.g., Elastic Search, Docker), logging tools (e.g., Logstash), and visualisation tools (e.g., Kibana) popular among microservice adopters. Depending on the logging and visualisation settings, different forms of data analytics can be carried on recorded values of these metrics (e.g., aggregation). Riemann [53] and IBM Bluemix [28] take an event-driven approach to monitoring by setting alerts on specific events or diagnosing event streams.

Overall, to answer Q1 related to the first evaluation goal, most of the inputs listed in Section 4.4 are unique compared to the tools discussed above. Regarding output, the examined tools do not explicitly perform any value-driven analysis or output value of granularity adaptation decisions on the recorded raw data like we do in our process. To answer Q2 related to the first evaluation goal, the output of our process is more informed despite being less user-friendly than other tools in some cases. On the other hand, the input to our process can be comparable in complexity relative to other tools.

Nevertheless, these tools are complementary rather than alternative to our process. They are all very suitable candidates to flexibly record a plethora of runtime evidence variables which can be used in our process. Other state-of-the-practice microservice orchestration platforms which can benefit from our value-driven analysis include Kubernetes [20] and Istio [5].

5.3 Results: Capturing Microservice Runtime Dynamics (Addressing Second Evaluation Goal)

To address the second evaluation goal as formulated using the GQM framework, we compare applying our process in the controlled experiment setting to applying traditional binomial real options analysis in the same setting. We calculate the total real option values which can be enabled by adapting and or retained by not adapting over 14 runtime iterations for each workload trend. We assume that the cost estimations are constant across the runtime iterations except for two cases where we manually update them. In this setting, we expect to see the following surprise trends:

- We expect the stepwise increase in workload to trigger a nasty surprise if the invocation duration is greater than 1 millisecond for strings less than 100 sentences long.
- We expect a good surprise if the invocation duration is less than 1 millisecond for strings larger than 100 sentences long.
- In the single spike workload, we expect to see the claim verified (i.e., invocation duration greater than 1 millisecond) when the review string is exceptionally long. Following that claim verification, we expect a good surprise to be triggered as the QoS improves (i.e., becomes less than 1 millisecond) after that spike. We expect to see a similar trend of claim verification followed by good surprises in the multiple spikes workload.

Table 2. Red cells = a nasty surprise is triggered; green cells = a good surprise; yellow cells = a surprise below the Bayesian surprise tolerance threshold T_s; grey cells = claim held true

	Monitoring Timeslot													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Review String Length/sentences	20	20	50	50	80	80	110	110	140	140	170	170	200	200
Invocation Duration/milliseconds	0.41	0.31	0.29	0.32	2.94	2.93	0.34	0.28	0.35	0.29	0.32	0.33	0.42	0.3
Bayesian surprise	0.194	0.06	0.147	0.07	0.246	0.147	0.07	0.66	0.07	0.07	0.07	0.07	0.07	0.194
Review String Length/sentences	20	20	50	50	80	80	110	110	140	140	1000	170	170	200
Invocation Duration/milliseconds	0.45	0.33	0.36	0.33	0.45	0.33	0.36	0.42	0.4	0.56	1.55	0.35	0.46	0.36
Bayesian surprise	0.195	0.08	0.147	0.08	0.246	0.147	0.06	0.06	0.07	0.66	0	0.195	0.06	0.05
Review String Length/sentences	20	20	50	50	80	80	110	1000	110	170	170	170	1120	170
Invocation Duration/milliseconds	0.53	0.38	0.41	0.46	0.51	0.4	0.43	1.55	0.35	0.46	0.35	0.28	2.9	0.35
Bayesian surprise	0.195	0.09	0.166	0.06	0.05	0.799	0.33	0	0.66	0.07	0.05	0.29	0	0.19

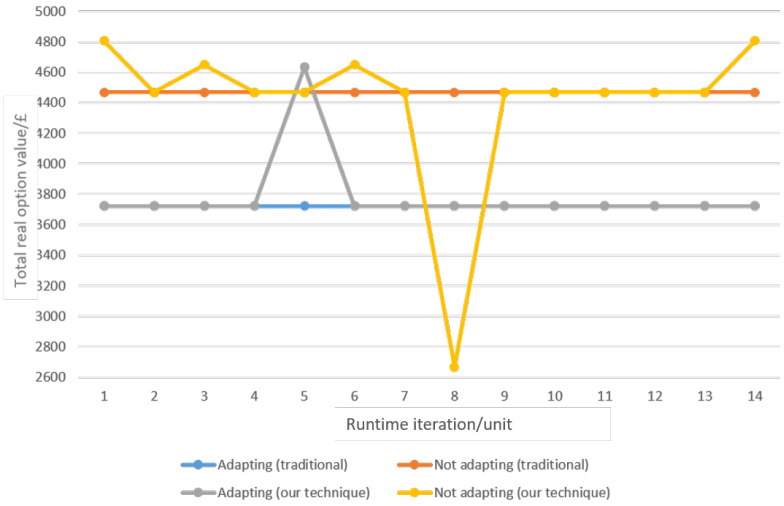


Fig. 5. Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with stepwise workload increase using our process against traditional analysis

Table 2 shows the data used to generate our contribution's outputs (Figures 5, 6, and 7). In Figures 5, 6, and 7, a point-wise comparison of our process against traditional binomial analysis reports fluctuation in the total real option values due to the real time calibration of the value enabled by adapting (or not). In contrast, traditional binomial analysis reports constant results for the total real option values. In the remainder of this subsection, we discuss the trends in *ROV* relative to those in REV for Figures 5, 6, and 7.

In Figure 5, runtime iteration 5 corresponds to a nasty surprise (0.246 in Table 2). The claim is violated since the invocation duration was greater than 1 millisecond for a review string less than 100 sentences long. It indicates that the system is behaving worse than initially believed. The triggered Bayesian surprise is nasty as per our first expectation regarding our contribution's behaviour. Therefore it increases the total real option value to be enabled by adapting, triggering a suggestion to adapt in this runtime iteration. It is up to the architect to take or to leave this suggestion. Our contribution provides the suggestion along with the objective justification for it in terms of potential added value.

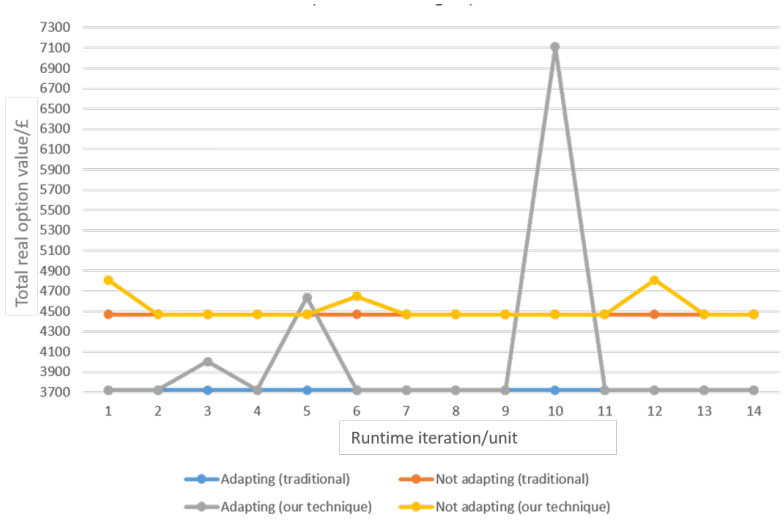


Fig. 6. Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with single spike workload using our process against traditional analysis

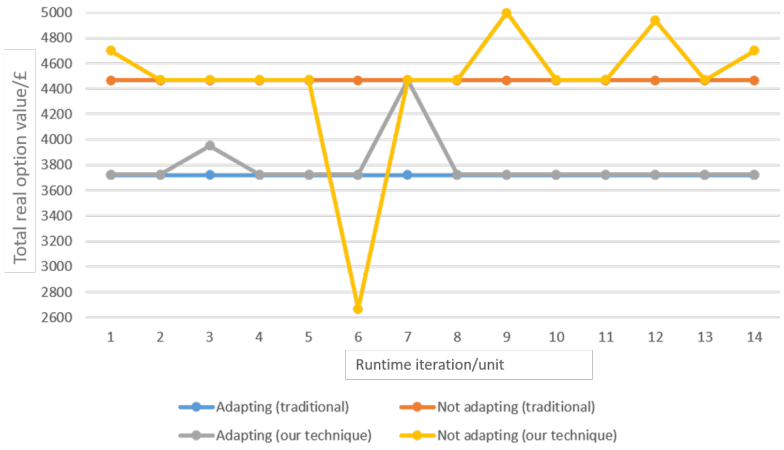


Fig. 7. Comparing real option values to be enabled by adapting and retained by not adapting granularity in the scenario with multiple outliers in workload using our process against traditional analysis

Therefore, our contribution provides a more informative suggestion regarding granularity adaptation compared to traditional binomial real options analysis. From iteration 9 to 13, the stepwise increase in workload does not trigger a significant surprise since shifts in the observed invocation duration are insignificant given the shifts in review string lengths.

In Figure 6, between iterations 9 and 10, the invocation duration has fluctuated even though the review string length has not changed. This combination gives significant information about the system's runtime behaviour triggering a nasty surprise above the tolerance threshold in runtime iteration 10. In iteration 11, a string 1000 sentences long is

injected and the invocation duration is greater than 1 millisecond. Therefore, the claim was verified in iteration 11 leading to zero Bayesian surprise, so the total real option values in our process and the traditional analysis were the same. The claim is then violated in iteration 12 triggering a good surprise (0.195) since the invocation duration was less than 1 millisecond for a review string longer than 100 sentences. This aligns with our second expectation regarding our evaluation process's behaviour. The Bayesian surprise in runtime iteration 12 therefore indicates that the situation in iteration 11 was a single spike and thereby our process calls for not adapting at iteration 12. This trend of claim verification followed by a good surprise aligns with our third expectation regarding our contribution's behaviour. Even though the traditional binomial real options analysis leads to the same verdict for that iteration, the total real option value corresponding to not adapting is higher in our process. This shows our process can more confidently argue for the verdict in terms of a higher total real option value corresponding to not adapting due to the runtime evidence records for iterations 11 and 12.

In Figure 7, between iterations 11 and 12 in the multiple spikes workload, the fluctuation of invocation duration triggered a good surprise at iteration 12. Runtime iterations 9 and 14 show good surprises (0.66 and 0.19, respectively). In iterations 9 and 14 the claim is violated since the invocation duration is less than 1 millisecond after exceptionally long strings are injected in iteration 8 and 13, respectively. Similar to Figure 6 and as per our expectation, there claim verification is followed by a good surprise in the multiple spikes workload as well.

However, the total real option value corresponding to not adapting in runtime iteration 14 is less than in iteration 9. This indicates that the smaller surprise in iteration 14 is due to less confidence about the behaviour of the system when long strings ("spikes") are injected frequently. This shows that our process can explicitly capture changes in the confidence as runtime evidence accumulates. The software architect can find this accumulated information useful in possible updates of the claim threshold. This can be done by calculating the mean review string length and invocation durations observed after a window of runtime iterations then mapping that to more relaxed claim thresholds.

Iteration 8 in Figure 5 and iteration 6 in Figure 7 show the impact of manual cost updates on the output of our process. Here we increase the maintenance cost of the un-adapted architecture when performing calculations using our process. The maintenance cost is kept constant when performing calculations using traditional binomial analysis. Despite the good surprises (0.66 in iteration 8 for stepwise workload increase and 0.799 for multiple spikes workload), our process promotes adapting for these iterations due to the high maintenance costs for the un-adapted architecture. This shows that our process can capture the runtime dynamics of cost as well. Traditional real options analysis might be able to only capture runtime dynamics if it is applied every time the costs are manually updated.

Overall, to answer Q3 related to the second evaluation goal, surprises in our approach are triggered when the monitored microservice(s) behave worse than expected within the end user workload threshold or when they behave better than expected beyond the end user workload threshold in each controlled experiment setting. Whenever surprises are triggered, the output of our process fluctuates, sometimes leading to a different verdict. The fluctuation in output incorporates rather than discards runtime evidence coming from the triggered surprises. Traditional binomial analysis is therefore less informed because it discards runtime evidence coming from the triggered surprises.

5.4 Proving Feasibility for Tool Support

We implement the AddedValueUpdate pseudocode into a Java tool to prove the feasibility of providing an interactive runtime decision support for microservice granularity adaptation grounded on our novel evaluation process. The tool first takes the inputs related to calculating Bayesian surprises (Figure 8), including location of the files where the runtime evidence variables values will be streamed. The tool then takes the inputs related to construct the binomial

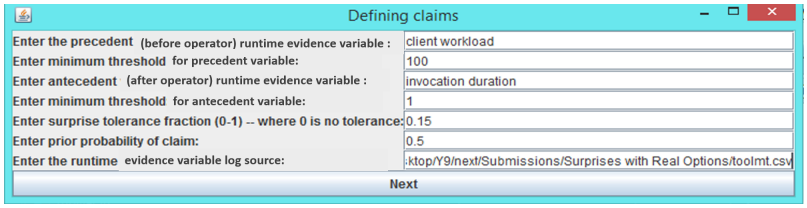


Fig. 8. Capturing inputs for calculating Bayesian surprises for the first iteration of our evaluation process used in the experiment in Section 5.3

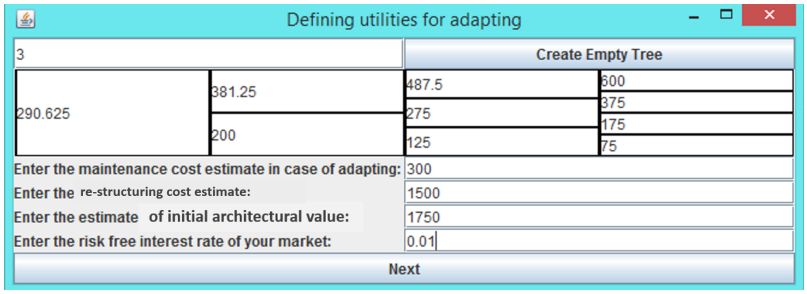


Fig. 9. Capturing the utility tree of adapting for the first iteration of our evaluation process used in the experiment in Section 5.3

trees for adapting (Figure 9) and not adapting the architecture (Figure 10) for the first runtime iteration. At the end of each iteration, the tool outputs a suggestion to adapt or not along with the real option value on which the suggestion is grounded (Figure 11).

Figures 8 to 11 also illustrate how even a primitive implementation of our process can abstract away internal calculations of AddedValueUpdate from the software architect. In particular, software architects do not need to have extensive knowledge of Bayesian surprises or binomial real option analysis to use our process. Nevertheless, they need to have sufficient knowledge of the sources from which they will obtain inputs to our process and to understand which source corresponds to which input field in the tool support. This knowledge can be provided to the architects either through help guides within the tool or by training them before using the tool. Regardless of the means of delivering the knowledge, its simplicity is more important. For example, the architect has to provide Bayesian surprise tolerance thresholds as input. Instead of explaining the intricate Bayesian surprise formulas, the architect can be asked to rank the claims in terms of relative importance. At the back-end of the tool then this can be translated to tolerance threshold percentages such that the higher the ranking of the claim the lower the corresponding tolerance threshold percentage. Another example is obtaining the maintenance cost estimates from the architect. To obtain these values it is important to show the architect the definitions of the terms corresponding to these values. Presenting term definitions will then help the architects decide on the suitable cost model to use for obtaining these values.

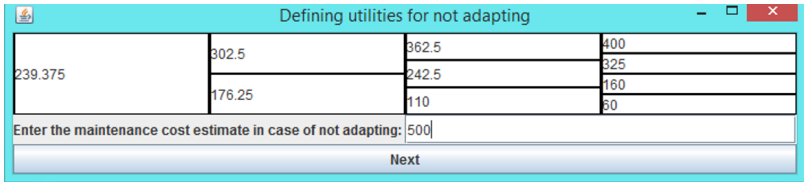


Fig. 10. Capturing the utility tree of not adapting for the first iteration of our evaluation process used in the experiment in Section 5.3

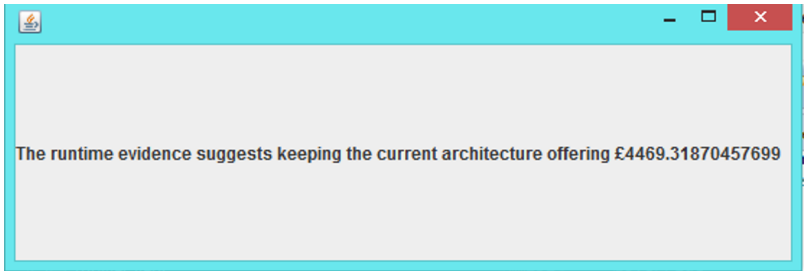


Fig. 11. Suggesting keeping the current architecture after the first iteration of our evaluation process

6 THREATS TO VALIDITY

We reflect on threats to validity of our evaluation results: internal, external, conclusion, and construct threats.

Internal: We acknowledge that the underlying techniques used in providing inputs to our process and setting up its usage greatly affect the extent to which our process is beneficial to microservice adopters. In our evaluation, we try to mitigate this issue by: 1) attempting to use suitable techniques for obtaining input values (e.g., CostHat and frequency tables) and, 2) varying the input values to stress test our process (e.g., injecting different workload trends to the monitored microservice).

External: Despite the evaluation being in a controlled Filmflix setting, we envision that our results are transferable to any microservice application domain for the following reasons: 1) the input values can hold different values depending on context, 2) our process is flexible regarding the input estimation techniques used, and 3) the rigour of our process can be changed depending on the context. To mitigate the bias of a controlled experiment setting, we stress test and varying inputs to our evaluation process. However, a plethora of industrial experiments is certainly desirable for independent evaluation and subject to future research due to space limitations.

Conclusion: Our monitoring has focused on chosen variables. The controlled experiment nature has stipulated controlling other inputs to our process apart from the monitored ones (i.e., workload and QoS attributes) or manually updated ones (i.e., maintenance cost) ensures that changes in the output are caused by changes in independent inputs. Though control may suffer bias, it is necessary to understand how our models are sensitive under various inputs that can be experienced in untypical usage scenarios.

In Section 5, we argue that Filmflix is of comparable size and scope to case studies used in relevant literature. Nevertheless, we acknowledge the threats to evaluation due to the controlled nature of the experiments. Though an effort was taken to analyse tighter than normal input parameters, we can see the need for further experimentation to cover systems from various microservice domains, scopes and scales. Such a wide range of uncontrolled experiments will factor into the evaluation observations, dynamism and uncertainties that may have been missed in our controlled

experiment. A larger experiment setting will provide us with more data points to further reflect on the strengths and weaknesses of the approach. Evaluation in a uncontrolled environment would expect to factor in more goals, questions and metrics that go beyond the controlled setups; these can be application dependent, domain- and case-specific. Our use of the GQM framework provides us with the flexibility to extend the evaluation, by considering more goals, questions and metrics.

Experimenting with larger case studies can pose manual overhead challenges when collating the input parameters required for the experiment. To reduce this overhead, our approach can be initially applied to a high-level model that mirrors the scale of the microservice architecture under investigation. The evaluation can then be conducted on sub-system within the microservice architecture of interest to the architect, while considering dependencies with other parts of the system.

Construct: We acknowledge that the suitability of binomial real options analysis for our contribution can be questioned due to the existence of other well-established architectural design decision evaluation processes in the literature. Section 3 argues that the target problem of this paper is best formulated as a real options problem. Hence, we found binomial real options analysis to be a suitable fit for evaluating the worth of pursuing granularity adaptation. We further acknowledge that the suitability of Bayesian surprises as a means to transit this analysis to runtime can be questioned due to limited application of this concept in the literature. However, our choice was motivated by striving for consistency in calculations and utilising runtime evidence. Binomial real options heavily utilise probabilities and Bayesian surprises also takes probabilities as input for calculation. Bayesian surprises relies on runtime evidence for assessing claims and our target problem requires runtime value updates. An interesting future research direction would be to update binomial trees at runtime using quantifications other than Bayesian surprises.

7 CONCLUSION AND FUTURE WORK

We reported on a novel formulation of the decision to pursue granularity adaptation as a real options problem, inspired by the theory of Baldwin and Clark [8]. We proposed a novel dynamic evaluation process that combines binomial real options analysis with Bayesian surprises to assess the added value of adapting granularity (or not) in microservices. Our work is the first to:

- Dynamically evaluate the potential added value under uncertainty related to pursuing microservice granularity adaptation.
- Enable dynamic application of binomial real options analysis with run-time update for its parameters; this is contracted to the traditional/ static design time usage.
- Apply the concept of Bayesian surprises in the context of binomial real options analysis and in microservice granularity adaptation.

We compared the output of a representative industrial microservice runtime monitoring tool — AWS Cloudwatch — against the output of our evaluation process. We further discussed the differences between the features provided by three other state-of-the-practice microservice monitoring tools — X-Pack, IBM Bluemix, and Riemann — versus those provided by our evaluation process. The results provide tangible evidence on the benefits of linking runtime granularity adaptation decisions to their added value under uncertainty. We also use a controlled experiment to show how our process goes beyond traditional application of real options by revealing the impact of runtime variations in workload on the added value.

In the short term future, our evaluation can be further strengthened. For example, we intend to experiment with: (1) observing rather than controlling workload and, (2) using alarms within AWS Cloudwatch to alert when claims are violated.

In the long term future, our proposed evaluation process opens up the path to answering another crucial related question: which granularity adaptation strategy (e.g., decomposing a microservice with unrelated features) is suitable at a certain point at runtime? To study this question, we will investigate how machine learning techniques can help in pruning the candidate space of granularity adaptation strategies. Pruning candidate strategies is particularly crucial to ensure practical use of granularity adaptation decision-making support for large-scale microservice case studies.

REFERENCES

[1] M. Ahmadvand and A. Ibrahim. 2016. Requirements Reconciliation for Scalable and Secure Microservice (De)composition. In *24th IEEE International Requirements Engineering Conference Workshops (REW)*. 68–73.

[2] Esra Alzaghoul and Rami Bahsoon. 2013. CloudMTD: Using real options to manage technical debt in cloud-based service selection. In *2013 4th International Workshop on Managing Technical Debt (MTD)*. 55–62. https://doi.org/10.1109/MTD.2013.6608680

[3] Martha Amram and Nalin Kulatilaka. 1999. *Real Options: Managing Strategic Investment in an Uncertain World.* Harvard Business School Press.

[4] Jayatirtha Asundi, Rick Kazman, and Mark Klein. 2001. *Using Economic Considerations to Choose Among Architecture Design Alternatives.* Technical Report CMU/SEI-2001-TR-035. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5785

[5] Istio Authors. [n.d.]. *Istio.* https://istio.io/

[6] AWS. 2017. *AWS Lambda Metrics.*

[7] AWS. 2017. *What is AWS Lambda?*

[8] Carliss Y. Baldwin and Kim B. Clark. 2000. *Design Rules, Volume 1 The Power of Modularity.* The MIT Press.

[9] Nelly Bencomo and Amel Belaggoun. 2014. A World Full of Surprises: Bayesian Theory of Surprise to Quantify Degrees of Uncertainty. In *36th Companion Proceedings of the International Conference on Software Engineering*. 460–463.

[10] Barry W. Boehm and Kevin J. Sullivan. 2000. Software Economics: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. 319–343.

[11] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* (1994), 528–532.

[12] Shawn Butler Mary Shaw Chris Scaffidi, Ashish Arora. 2005. A Value-Based Approach to Predicting System Properties from Design. In *ACM SIGSOFT Software Engineering Notes*. 1–5.

[13] Neil A. Chriss. 1997. *The Black-Scholes and Beyond and the Black-Scholes and Beyond Interactive Toolkit: A Step-By-Step Guide to In-Depth Option Pricing Models.* McGraw-Hill Trade.

[14] Zhamak Dehghani. 2015. Zhamak Dehghani Real World Microservices: Lessons from the Frontline. https://youtu.be/hsoovFbpAoE.

[15] Sergio del Amo Caballero. 2018. Micronaut Tutorial: How to Build Microservices with this JVM-based Framework. https://www.infoq.com/articles/micronaut-tutorial-microservices-jvm.

[16] Hakan Erdogmus. 2001. Management Of License Cost Uncertainty In Software Development: A Real Options Approach. In *5th Annual Conference on Real Options: Theory Meets Practice.*

[17] Hakan Erdogmus and John Favaro. 2002. Keep Your Options Open: Extreme Programming and Economics of Flexibility. In *In*. Addison Wesley, 503–552.

[18] H. Erdogmus and J. Vandergraaf. 1999. Quantitative approaches for assessing the value of COTS-centric development. In *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*. 279–290. https://doi.org/10.1109/METRIC.1999.809749

[19] L. Florio and E. D. Nitto. 2016. Gru: An Approach to Introduce Decentralized Autonomic Behavior in Microservices Architectures. In *IEEE International Conference on Autonomic Computing (ICAC)*. 357–362.

[20] The Linux Foundation. [n.d.]. *Production-Grade Container Orchestration.* https://kubernetes.io/

[21] Martin Fowler. 2004. Strangler Application. https://www.martinfowler.com/bliki/StranglerApplication.html.

[22] Jonas Fritzsch. 2018. *From Monolithic Applications to Microservices Guidance on Refactoring Techniques and Result Evaluation.* Master's thesis. Reutlingen University.

[23] Andrea Gamba. 2003. Real options valuation: A Monte Carlo approach. (2003).

[24] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. 2017. Towards Recovering the Software Architecture of Microservice-Based Systems. In *IEEE International Conference on Software Architecture Workshops (ICSAW)*. 46–53.

[25] Sara Hassan, Rami Bahsoon, and Nelly Bencomo. 2015. Minimizing Nasty Surprises with Better Informed Decision-Making in Self-Adaptive Systems. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 134–145.

[26] N. Herzberg, C. Hochreiner, O. Kopp, and J. Lenhard. 2018. Challenges of Microservices Architecture: A Survey on the State of the Practice. In *10th ZEUS Workshop (ZEUS)*.

[27] Tom Huston. [n.d.]. What is Microservices Architecture? https://smartbear.com/learn/api-design/what-are-microservices/

[28] IBM. 2017. *IBM BlueMix Container Service- Monitoring and Logging*.

[29] Investopedia. [n.d.]. Compound Option. https://www.investopedia.com/terms/c/compoundoption.asp.

[30] Rick Kazman, Mark Klein, and Paul Clements. 2000. *ATAM: Method for Architecture Evaluation*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

[31] Gabor Kecskemeti, Attila Kertesz, and Attila Csaba Marosi. 2017. Towards a Methodology to Form Microservices from Monolithic Ones. In *Euro-Par 2016: Parallel Processing Workshops*. 284–295.

[32] G. Kecskemeti, A. C. Marosi, and A. Kertesz. 2016. The ENTICE approach to decompose monolithic services into microservices. In *International Conference on High Performance Computing Simulation (HPCS)*. 591–596.

[33] Fondazione Brundo Kessler. 2014. ASTRO-CAptEvo. Online. Retrieved 5 August 2017 from http://das.fbk.eu/astro-captevo

[34] Sander Klock, Jan Martijn E. M. Van Der Werf, Jan Pieter Guelen, and Slinger Jansen. 2017. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In *2017 IEEE International Conference on Software Architecture (ICSA)*. 11–20. https://doi.org/10.1109/ICSA.2017.38

[35] L. Krause. 2015. *Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns*. Lucas Krause.

[36] N. Kulkarni and V. Dwivedi. 2008. The Role of Service Granularity in a Successful SOA Realization A Case Study. In *IEEE Congress on Services - Part I*. 423–430.

[37] P. Leitner, J. Cito, and E. Stöckli. 2016. Modelling and Managing Deployment Costs of Microservice-Based Cloud Applications. In *9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. 165–174.

[38] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. 2016. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. *CoRR* abs/1605.03175 (2016). arXiv:1605.03175 http://arxiv.org/abs/1605.03175

[39] James Lewis. 2013. Micro Services: Java, the Unix Way. http://www.infoq.com/presentations/Micro-Services.

[40] Logstash. 2017. *Monitoring Logstash*.

[41] John D. McGregor, David P. Gluch, and Peter H. Feiler. 2017. Analysis and Design of Safety-critical, Cyber-Physical Systems. *Ada Lett.* 36, 2 (2017), 31–38.

[42] Karl Meinke and Peter Nycander. 2015. Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection. In *Software Engineering and Formal Methods*. 3–10.

[43] Ola Mustafa, Jorge Marx Gómez, Mohamad Hamed, and Hergen Pargmann. 2018. GranMicro: A Black-Box Based Approach for Optimizing Microservices Based Applications. In *From Science to Society*. 283–294.

[44] Ola Mustafa and Jorge Marx Gómez. 2017. Sustainable approach for improving microservices based web application. In *Sustainability Dialogue: International Conference on Sustainability and Environmental Management*.

[45] S.C. Myers. 1976. *Modern Developments in Financial Management*. Praeger.

[46] Sam Newman. 2015. *Building Microservices*. O'Reilly Media.

[47] Michael Nygard. 2007. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf.

[48] Ipek Ozkaya, Rick Kazman, and Mark Klein. 2007. *Quality-Attribute-Based Economic Valuation of Architectural Patterns*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

[49] L. H. G. Paucar and N. Bencomo. 2016. The Reassessment of Preferences of Non-functional Requirements for Better Informed Decision-Making in Self-Adaptation. In *24th IEEE International Requirements Engineering Conference Workshops (REW)*. 32–38.

[50] Christian Posta. 2017. The Hardest Part of Microservices: Calling Your Services. http://blog.christianposta.com/microservices/the-hardest-part-of-microservices-calling-your-services/.

[51] Katharina Probst and Justin Becker. 2016. Engineering Trade-Offs and The Netflix API Re-Architecture. http://techblog.netflix.com/2016/08/engineering-trade-offs-and-netflix-api.html.

[52] C. Richardson. 2018. *Microservice Patterns*. Manning Publications Company.

[53] Riemann. 2017. *Riemann - Concepts*.

[54] Casey Rosenthal. 2016. GOTO 2016 — Chaos & Intuition Engineering at Netflix — Casey Rosenthal. https://www.youtube.com/watch?v=Q4nniyAarbs.

[55] Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited.

[56] J. P. Guelen S. Klock, J. M. E. M. van der Werf and S. Jansen. 2017. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In *IEEE International Conference on Software Architecture (ICSA)*. 11–20.

[57] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. 2005. Using Dependency Models to Manage Complex Software Architecture. In *20th Annual Proceedings of the ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. 167–176.

[58] D. I. Savchenko, G. I. Radchenko, and O. Taipale. 2015. Microservices validation: Mjolnirr platform case study. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 235–240.

[59] Vibha Sazawal and Nikita Sudan. 2009. Modeling Software Evolution with Game Theory. In *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*. 354–365.

[60] Eduardo S. Schwartz. 1977. The valuation of warrants: Implementing a new approach. *Journal of Financial Economics* 4, 1 (1977), 79 – 93.

[61] M. Schwartz. 2016. *The Art of Business Value:*. IT Revolution Press. https://books.google.co.uk/books?id=gykhDgAAQBAJ

[62] Yuanyuan Song. 2007. Adaptation Hiding Modularity for Self-Adaptive Systems. In *Companion to the Proceedings of the 29th International Conference on Software Engineering*. 87–88.

[63] Kevin J Sullivan, Prasad Chalasani, Somesh Jha, and Vibha Sazawal. 1999. Software Design as an Investment Activity: A Real Options Perspective. *Real options and business strategy: Applications to decision making* 10 (1999), 215–262.

[64] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, and Ben Hallen. 2001. The Structure and Value of Modularity in Software Design. In *8th Proceedings of the European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 99–108.

[65] D. Taibi and V. Lenarduzzi. 2018. On the Definition of Microservice Bad Smells. *IEEE Software* 35, 3 (2018), 56–62.

[66] L. Trigeorgis and P.F.L. Trigeorgis. 1996. *Real Options: Managerial Flexibility and Strategy in Resource Allocation.* PAPERBACKSHOP UK IMPORT.

[67] Tim Wagner. 2015. Microservices without the Servers. https://aws.amazon.com/blogs/compute/microservices-without-the-servers/

[68] E. Wolff. 2016. *Microservices: Flexible Software Architecture.* Pearson Education.

[69] Liguo Yu and Srini Ramaswamy. 2008. Improving Modularity by Refactoring Code Clones: A Feasibility Study on Linux. *SIGSOFT Softw. Eng. Notes* 33, 2 (2008), 1–5.

Appendix A STEP-BY-STEP EXAMPLE OF PROPOSED DYNAMIC EVALUATION PROCESS

In this section, we show a step-by-step application for a single iteration of Alg. 1) — AddedValueUpdate — to Filmflix's initial architecture. We list input values to the pseudocode (expanding on Section 4.1), specify how they are calculated where necessary, plug the input values into Alg. 1, and show how the pseudocode leads to the process's output.

A.1 Inputs

- *Microservice:* MovieReview
- *QoS attribute (a runtime evidence variable):* invocation duration of MovieReview
- *Workload (a runtime evidence variable):* review string submitted by the end user
- *QoS attribute threshold:* 1 millisecond
- *End user workload threshold:* 100 sentences long
- *Bayesian surprise tolerance threshold:* 15%.
- *Claim posterior probability:* The posterior probability of the claim holding true (*P(Claim | REV)* in Eq. (5.1)) is derived from frequency tables.
- *RC:* £1500
- *MC adapted architecture:* £300
- *MC un-adapted architecture:* £500
- *$Value_0$:* £1750
- *Trees:*

Table 3. Utility tree showing the likely utility value of improving the invocation duration over the following three runtime iterations when granularity level of *MovieReview* is adapted

0	1	2	3
			G
		D	600
	B	487.5	H
A	381.25	E	375
290.625	C	275	I
	200	F	175
		125	J
			75

- F_t*:* three iterations
- $Q_{ri\ name}$*:* invocation duration
- T_w*:* 100 sentences
- T_q*:* 1 millisecond
- P_{claim}*:* 0.5
- T_s*:*15%

Table 4. Utility tree showing the likely utility value of improving the invocation duration over the following three runtime iterations when granularity level of *MovieReview* is kept unchanged

0	1	2	3
			G
		D	400
	B	362.5	H
A	302.5	E	325
239.375	C	242.5	I
	176.25	F	160
		110	J
			60

A.2 Process

(1) Claim: [W_{ri} > 100 sentences ↔ Q_{ri} > 1 millisecond
(2) ri=0
(3) Construct Real Options Trees for Adapting (Figure 12)
(4) Construct Real Options Trees for not Adapting (Figure 13)
(5) Solicit and Monitor Q=0.41 milliseconds and W=20 sentences
(6) Calculate S^{claim} (using Eq. 9) and $T_{s\ exact}$ using (Eq. 10). $P(Claim|REV)$ needs to calculated before calculating S_{claim}.

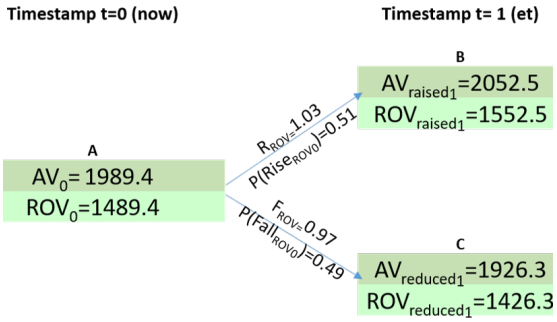
$$S_{\text{claim}} = P(Claim|REV) * \log \frac{P(Claim|REV)}{0.5} = 0.194 \tag{12}$$

$$T_S^{exact} = 0.15 * (0.847 - 0.194) = 0.098 \tag{13}$$

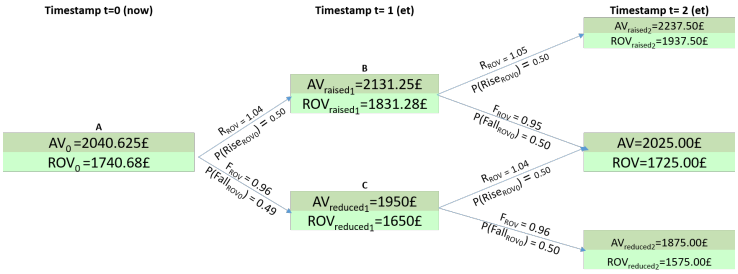
(7) Execute conditional: (Q_{ri} < $Q_{ri\text{-}1}$) AND (W > T_w)
$P(Rise_{\text{ROV}}{}^{\text{not adapting}}) * 0.194/0.098$
(8) Re-calculate $ROV_{total}{}^{adapting}$ and $ROV_{total}{}^{\text{not adapting}}$ using updated probabilities
(9) Execute Else clause: $Suggestion_{ri}$=No

A.3 Output

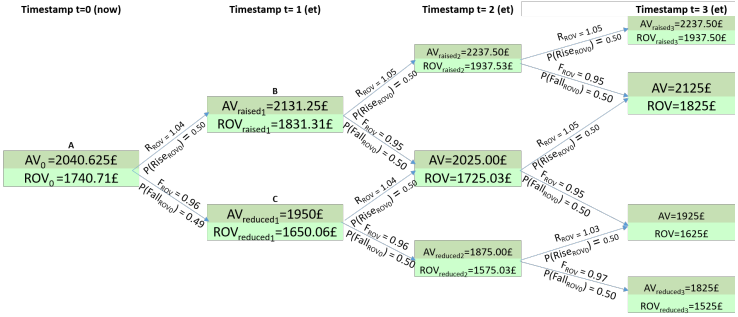
$ROV_{total}^{adapting}$, $ROV_{total}^{notadapting}$, $Suggestion_{ri}$



(a) Intermediate binomial real option tree for adapting if the real option is exercised after the first timestamp before the option expires

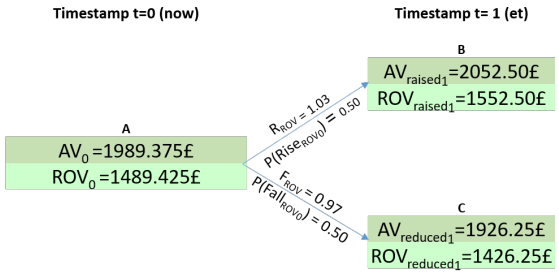


(b) Intermediate binomial real option tree for adapting if the real option is exercised after the second timestamp before the option expires

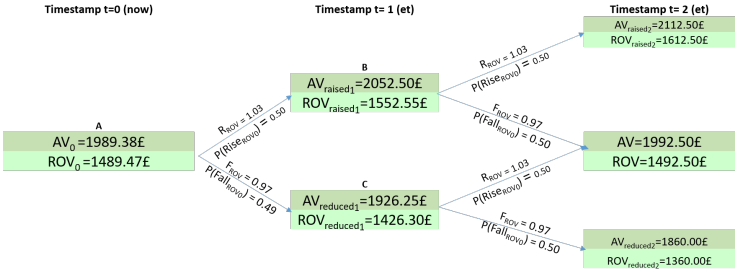


(c) Intermediate binomial real option tree for adapting if the real option is exercised after the third timestamp just before the option expires

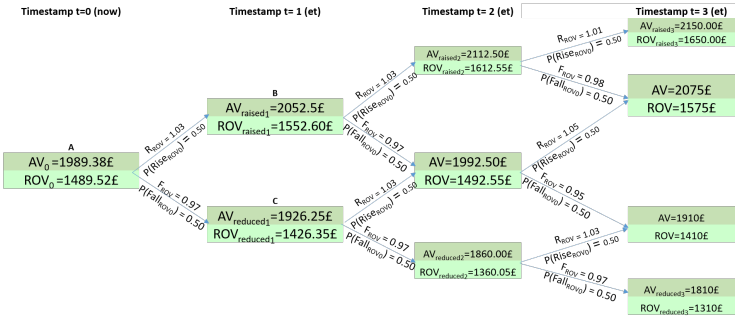
Fig. 12. Creating binomial real options analysis tree for adapting granularity of *MovieReview* after the first runtime iteration of our evaluation process



(a) Intermediate binomial real option tree for not adapting *MovieReview* if an embedded real option is exercised after the first timestamp before the option expires



(b) Intermediate binomial real option tree for not adapting *MovieReview* if an embedded real option is exercised after the second timestamp before the option expires



(c) Intermediate binomial real option tree for not adapting *MovieReview* if an embedded real option is exercised after the third timestamp before the option expires

Fig. 13. Creating binomial real options analysis tree for not adapting granularity of *MovieReview* after the first runtime iteration of our evaluation process