The Journal of Systems & Software 188 (2022) 111275

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Quantitative verification with adaptive uncertainty reduction[☆]

Naif Alasmari [a], Radu Calinescu [a,*], Colin Paterson [a], Raffaela Mirandola [b]

[a] Department of Computer Science, University of York, UK
[b] Politecnico di Milano, Italy

ARTICLE INFO

Article history:
Received 17 March 2021
Received in revised form 23 January 2022
Accepted 16 February 2022
Available online 22 February 2022

Keywords:
Quantitative verification
Probabilistic model checking
Confidence intervals
Uncertainty reduction
Nonfunctional requirements
Unit testing

ABSTRACT

Stochastic models are widely used to verify whether systems satisfy their reliability, performance and other nonfunctional requirements. However, the validity of the verification depends on how accurately the parameters of these models can be estimated using data from component unit testing, monitoring, system logs, etc. When insufficient data are available, the models are affected by epistemic parametric uncertainty, the verification results are inaccurate, and any engineering decisions based on them may be invalid. To address these problems, we introduce VERACITY, a tool-supported iterative approach for the efficient and accurate verification of nonfunctional requirements under epistemic parameter uncertainty. VERACITY integrates confidence-interval quantitative verification with a new *adaptive uncertainty reduction* heuristic that collects additional data about the parameters of the verified model by unit-testing specific system components over a series of verification iterations. VERACITY supports the quantitative verification of discrete-time Markov chains, deciding which components are to be tested in each iteration based on factors that include the sensitivity of the model to variations in the parameters of different components, and the overheads (e.g., time or cost) of unit-testing each of these components. We show the effectiveness and efficiency of VERACITY by using it for the verification of the nonfunctional requirements of a tele-assistance service-based system and an online shopping web application.

1. Introduction

Many software systems must satisfy dependability, performance, cost and other requirements not directly related to the functionality they provide. These requirements are termed *nonfunctional requirements* (Ameller et al., 2019; Chung et al., 2012), and their verification needs to consider the stochastic nature of software characteristics such as inputs, workloads, timeouts and failures. As such, stochastic models ranging from Markov chains (Calinescu et al., 2011; Gallotti et al., 2008) and probabilistic automata (Krka et al., 2010; Johnson et al., 2013) to stochastic Petri nets (Marsan et al., 1998; Perez-Palacin et al., 2012) are widely used to perform this verification. However, ensuring that stochastic models are sufficiently accurate to support this verification is very challenging. While the structure of the models can be extracted from the actual code (Filieri et al., 2013) or from software artefacts such as activity diagrams (Gallotti et al., 2008; Calinescu et al., 2013), their parameters (e.g., probabilities, timing and other quantitative information) are affected by uncertainty.

These parameters need to be estimated using data obtained, for instance, from testing the system components individually, or (for systems already in use) from system logs. Point estimators such as the mean of the observed parameter values are typically used for this purpose. However, the point estimation of the uncertain model parameters produces imprecise verification results, and risks causing invalid engineering decisions (Calinescu et al., 2015; Paterson and Calinescu, 2020), especially when only few observations are available. In mature subjects like medicine (Akobeng, 2008; Kirkwood and Sterne, 2010) and in established engineering disciplines like civil (Benjamin and Cornell, 2014) and mechanical (Aughenbaugh and Paredis, 2006) engineering, this risk is deemed unacceptable, and it is mitigated by computing *confidence intervals* for the model parameters and the verified properties (du Prel et al., 2009; Gardner and Altman, 1986). In contrast, this risk is rarely considered in software performance and dependability engineering. Instead, the research in this area focuses on devising new techniques, tools and applications for the verification of stochastic models, under the strong assumption that using point estimates for the model parameters is sufficiently accurate.

To address the risk of invalid decisions associated with this assumption, we introduce VERACITY,[1] a tool-supported approach for the quantitative verification of discrete-time Markov chains

[☆] Editor: W. Eric Wong.
[*] Corresponding author.
E-mail address: Radu.Calinescu@york.ac.uk (R. Calinescu).

[1] quantitative VERification with Adaptive unCertaInTY reduction.

https://doi.org/10.1016/j.jss.2022.111275
0164-1212/© 2022 Elsevier Inc. All rights reserved.

under epistemic parametric (i.e., transition probability) uncertainty. Uncertainty is termed *epistemic* when it is due to insufficient data (and therefore reducible by gathering additional data), and *aleatory* when it is intrinsic to the analysed system (and therefore irreducible) (Perez-Palacin and Mirandola, 2014).

VERACITY builds on our previous research on computing confidence intervals for the reliability, performance and other nonfunctional properties of a system (Calinescu et al., 2015, 2016). This computation uses a *parametric discrete-time Markov chain* (i.e., a discrete-time Markov chain with unknown state transition probabilities) that models the system behaviour, and observations of the system behaviour available from component unit testing, runtime monitoring or system logs. However, when insufficient observations are available, these confidence intervals are too wide to verify whether nonfunctional requirements that impose constraints on such properties are satisfied. As an example, given too few observations, the 99% confidence interval for the probability that user requests are handled successfully by a web server may be [0.76, 0.98],[2] which is too wide for verifying the requirement *'The web server shall handle user requests with a success probability of at least 0.92 at 99% confidence level.'* To handle this frequently encountered problem efficiently, VERACITY obtains additional observations by unit-testing specific system components over a series of *adaptive uncertainty reduction* iterations. The components tested in each iteration are decided using a heuristic that takes into account multiple factors. These factors are detailed later in the paper, and include the sensitivity of the nonfunctional properties to variations in the parameters associated with different components, and the overheads (e.g., time or cost) of testing each of these components.

VERACITY supports both the verification of new system designs, and the verification of planned updates to existing systems. Using VERACITY to decide whether a new system should be deployed or not involves applying our approach with few or no initial observations of the system parameters. Multiple uncertainty reduction iterations are typically required to acquire sufficient observations of the parameters of the system and to reach a decision in this case. In contrast, when deciding whether updated versions of specific system components should be adopted, VERACITY can exploit a large number of initial observations of the parameters associated with the components not being updated. Accordingly, fewer uncertainty reduction iterations are typically necessary in this case, primarily to acquire observations of the parameters associated with the updated components.

VERACITY relies on the possibility to test the components of a system individually. Such unit testing of software components is widely used in software development (Daka and Fraser, 2014). For certain types of software systems, the individual testing of their components is also feasible after deployment. As an example, the third-party services used by a service-based system can be invoked independently at any stage of the software development life cycle.

The contributions of the paper are threefold. First, we introduce a new heuristic for the efficient reduction of epistemic parametric uncertainty of Markov chains used in dependability and performance software engineering. Second, we present a new approach that integrates this heuristic with a recently proposed method for formal verification with confidence intervals (Calinescu et al., 2015, 2016), and a tool that implements the approach, automating the verification of nonfunctional requirements under parametric uncertainty. Finally, we present extensive experimental results showing: (i) the effectiveness of the VERACITY verification approach during initial software development and software updating; and (ii) the efficiency of the VERACITY uncertainty reduction compared to uncertainty reduction by uniformly testing all the components of the system under verification (SUV).

We organised the remainder of the paper as follows. Section 2 defines the probabilistic model checking and formal verification with confidence concepts and notation required to present VERACITY. Section 3 introduces a motivating example that we then use to present our quantitative verification approach in Section 4. Sections 5 and 6 describe the tool support we implemented for our approach, and the case studies we carried out to evaluate VERACITY, respectively. Finally, we discuss related work in Section 7, and we conclude with a brief summary and we suggest directions for future work in Section 8.

2. Preliminaries

2.1. Parametric Markov chains

Definition 1. A discrete-time Markov chain is a tuple

$$M = (S, s_0, \mathbf{P}, L), \tag{1}$$

where: S is a finite set of states; $s_0 \in S$ is the initial state; $\mathbf{P} : S \times S \to [0, 1]$ is a transition probability matrix such that, for any states $s, s' \in S$, $\mathbf{P}(s, s')$ is the probability of transition from s to s' and, for any $s \in S$, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$; and $L : S \to 2^{AP}$ is a function that maps every state $s \in S$ to those elements of an atomic proposition set AP that hold in state s.

To extend the range of properties that can be verified using discrete-time Markov chains, their states and/or transitions are often annotated with non-negative quantities termed *rewards*.

Definition 2. A *reward structure* over a discrete-time Markov chain M is a pair of functions (ρ, ι) that map the states and state transitions of M to non-negative quantities called *rewards*: $\rho : S \to [0, \infty)$ and $\iota : S \times S \to [0, \infty)$.

For Markov chains used in software performance and dependability engineering, the states may correspond to different SUV configurations, to different operations being executed, to different outcomes of these executions, etc. In these models, the rewards may specify expected execution times, resource use and other quantitative characteristics of the operations carried out by the SUV. Finally, the continuous variables used to define the unknown transition probabilities and rewards represent parameters of the SUV components.

Definition 3. A *parametric (discrete-time) Markov chain* is a discrete-time Markov chain comprising one or several unknown state transition probabilities and/or rewards that are specified as rational functions (i.e., as fractions whose numerators and denominators are polynomial functions, e.g., $1-p$ or $(1-p_1)/p_2$ (The European Mathematical Society, 2020)) over a set of continuous variables (Daws, 2005).

The continuous variables used to specify the transition probabilities and/or rewards of parametric Markov chains correspond to parameters of the SUV.

2.2. Probabilistic computation tree logic

To verify the nonfunctional requirements of a system modelled by a parametric Markov chain, these requirements are

[2] In the case when no observations are available (corresponding to the common scenario of a system that has not yet been tested), this confidence interval is [0, 1].

expressed in rewards-extended (Andova et al., 2004) probabilistic computation tree logic (PCTL) (Ciesinski and Größer, 2004; Hansson and Jonsson, 1994) with the syntax:

$$\begin{aligned}\Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}[\Psi] \\ \Psi &::= \mathrm{X}\ \Phi; \mid \Phi\ \mathrm{U}^{\leq k}\ \Phi; \mid \Phi\ \mathrm{U}\ \Phi \\ \Theta &::= \mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{F}\ \Phi] \mid \mathcal{R}_{\bowtie r}[\mathrm{S}]\end{aligned} \tag{2}$$

where Φ is a *state formula*, Ψ a *path formula*, Θ a *reward state formula*, $k \in \mathbb{N}$ a timestep bound, $\bowtie\, \in \{\leq, <, \geq, >\}$ a relational operator, $p \in [0, 1]$ a probability bound, $r \geq 0$ a reward bound, and $a \in AP$ an atomic proposition.

The PCTL semantics is defined using a satisfaction relation $\models$. Given a state s of a Markov chain M, $s \models \Phi$ means "Φ holds in state s", and we have: always $s \models true$; $s \models a$ iff $a \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$; and $s \models \mathcal{P}_{\bowtie p}[\Psi]$ iff the probability x that paths starting at state s (i.e., sequence of states $s_1 s_2 s_3 \ldots$ such that $s_1 = s$ and $\forall i > 0 : \mathbf{P}(s_i, s_{i+1}) > 0$) satisfy the path property Ψ satisfies $x \bowtie p$. The *next formula* $\mathrm{X}\ \Phi$ holds for a path if Φ is satisfied in the next state on the path; the *bounded until formula* $\Phi_1\, \mathrm{U}^{\leq k}\, \Phi_2$ holds for a path iff Φ_1 holds in the first $i < k$ path states and Φ_2 holds in the $(i + 1)$th path state; and the *unbounded until formula* $\Phi_1\, \mathrm{U}\, \Phi_2$ removes the bound k from the time-bounded until formula. Finally, the four reward state formulae Θ from (2) use the reward operator $\mathcal{R}$ to verify whether the expected reward x: at timestep k; accumulated up to timestep k; accumulated to reach a state that satisfies Φ; and at steady state, respectively, satisfies $x \bowtie r$. For a detailed description of the PCTL semantics, see Andova et al. (2004), Ciesinski and Größer (2004) and Hansson and Jonsson (1994).

Additionally, the notation $\mathrm{F}^{\leq k}\ \Phi \equiv \mathrm{true}\ \mathrm{U}^{\leq k}\ \Phi$ and $\mathrm{F}\ \Phi \equiv \mathrm{true}\ \mathrm{U}\ \Phi$ is used when the left-hand side of a bounded until and until formula, respectively, is *true*; and $\mathcal{P}_{=?}[\ \cdot\ ]$ and $\mathcal{R}_{=?}[\ \cdot\ ]$ are used to denote the value of the probability and expected reward from a PCTL state and reward state formula, respectively.

2.3. Formal verification with confidence intervals

Formal verification with confidence intervals (Calinescu et al., 2015) is a mathematically based technique for the computation of confidence intervals for nonfunctional properties of systems with stochastic behaviour. Given a parametric Markov chain $M = (S, s_0, \mathbf{P}, L)$ that models the behaviour of a SUV, a PCTL formula $\mathcal{P}_{=?}[\cdot]$ or $\mathcal{R}_{=?}[\cdot]$ corresponding to a nonfunctional property of the SUV, and a confidence level $\alpha \in (0, 1)$, the technique computes an α confidence interval for the property.

To perform this computation, the technique starts by calculating confidence intervals for the parameters of the Markov chain by using observations of the outgoing transitions from all states with unknown outgoing transition probabilities. Assuming that $Z \subseteq S$ is the subset of these states, the required observations are provided by a function

$$O : Z \times S \to \mathbb{N} \tag{3}$$

that maps each pair of states $(z, s) \in Z \times S$ to the number $O(z, s)$ of times the transition from z to s has been observed, within the given observation time. We note that $O(z, s) = 0$ for the states $s \in S$ for which no transition from z to s was observed. This may be the case even when such transitions are possible (e.g., because they correspond to rare SUV events), and the potential lack of these observations is the very reason for using confidence intervals instead of point estimates in the formal verification. Given such observations, the confidence interval computation is done in three stages. In the first stage, a confidence interval is calculated for each parameter of the Markov chain. In the next stage, parametric model checking is used to obtain a closed-form expression (i.e., a mathematical expression containing only the basic arithmetic operators $+$, $-$, $\times$ and $/$, and exponent, e.g., $\frac{p_1(1-p_2)^2}{1-p_3p_4}$) for the nonfunctional property.[3] This expression is a rational function over the SUV parameters, and is a byproduct of the technique exploited by our VERACITY approach. Finally, in the third stage, the parameter confidence intervals and the property expression are used to establish the confidence interval for the nonfunctional property of interest. For a detailed description of the technique and of a model checker that implements it see Calinescu et al. (2015, 2016), respectively. This model checker supports non-nested PCTL properties $\mathcal{P}_{=?}[\cdot]$, and all types of PCTL reward properties.

The number of available observations and the confidence level α used by the technique influence the width of the confidence interval. Thus, few observations and large α values yield wide confidence intervals that may contain the lower/upper bound that a nonfunctional requirement specifies for the property. In this case, verifying whether the requirement is satisfied or not is impossible, and additional observations need to be collected to allow the computation of a narrower confidence interval that does not contain the bound.

3. Motivating example

To motivate our VERACITY approach, we use a tele-assistance service-based system (TAS) introduced in Baresi et al. (2007). TAS aims to support a patient suffering from a chronic condition in the comfort of their home by using: (i) a set of vital-sign monitoring sensors mounted on a medical device worn by the patient; and (ii) remote assistance services offered by emergency, medical and pharmacy service providers.

The workflow implemented by the TAS system is modelled by the parametric Markov model from Fig. 1.[4] Periodically, the patient's vital signs are measured by the wearable device (a workflow step whose completion is modelled by the Markov model transition from the initial state s_1 to state s_2), and a third-party medical analysis service is invoked to analyse them in conjunction with the patient's medical record (state s_2). This invocation may succeed (transition $s_2 \to s_4$) or fail (transition $s_2 \to s_3$). Depending on the results of this analysis (state s_4), TAS may confirm that the patient is fine (transition $s_4 \to s_9$), may invoke a pharmacy service to request the delivery of different medication to the patient's home (state s_5, followed by a transition to s_9 if the invocation succeeds or to s_7 otherwise), or may invoke an alarm service (state s_6, followed by a transition to s_9 if the invocation succeeds or to s_8 otherwise). The invocation of the alarm service is also triggered when the patient presses a panic button on the wearable device (modelled by the transition $s_1 \to s_6$), and results in a medical team being dispatched to provide emergency assistance to the patient. Once the TAS system is done with executing this workflow (state s_9), it may return to the state in which it awaits further requests (state s_1) or it may reach the end of its deployment (state s_{10}, whose self-loop of probability 1 shows that no further transition to other Markov model states is possible). We assume that the operational profile of the system is known (e.g., from previous deployments) and is given by the probabilities annotating the outgoing transitions from states s_1, s_4 and s_9 from Fig. 1.

[3] Parametric model checking is supported by a growing number of model checkers, including PARAM (Hahn et al., 2010), PRISM (Kwiatkowska et al., 2011), Storm (Dehnert et al., 2017) and ePMC/fPMC (Calinescu et al., 2021; Fang et al., 2021).

[4] As described in Calinescu and Rafiq (2013) and Ghezzi et al. (2013) and summarised in the Appendix, discrete-time Markov models for software systems can be derived from established software engineering models such as UML activity diagrams.

Table 1
Nonfunctional requirements for the TAS system.

ID	Requirement	PCTL formula
R1	The probability that an alarm failure ever occurs during the lifetime of the TAS system shall be below 0.26.	$\mathcal{P}_{<0.26}[\mathrm{F}\ \mathit{alarmFail}]$
R2	The probability that the handling of a request by the TAS workflow ends with a service failure shall be below 0.04.	$\mathcal{P}_{<0.04}[\neg \mathit{done}\ \mathrm{U}\ \mathit{serviceFail}]$
R3	The probability that an invocation of the medical analysis service is followed by an alarm failure shall be below 0.0003.	$\mathcal{P}_{<0.0003}[\neg \mathit{done}\ \mathrm{U}\ \mathit{alarmFail}\{\mathit{analysis}\}]$[a]

[a]We adopt the extended PCTL syntax of the PRISM model checker (Kwiatkowska et al., 2011) to specify that this PCTL formula should be verified for the Markov chain state for which the atomic proposition *analysis* holds (i.e., state s_2) instead of the initial state s_0, which is the default.

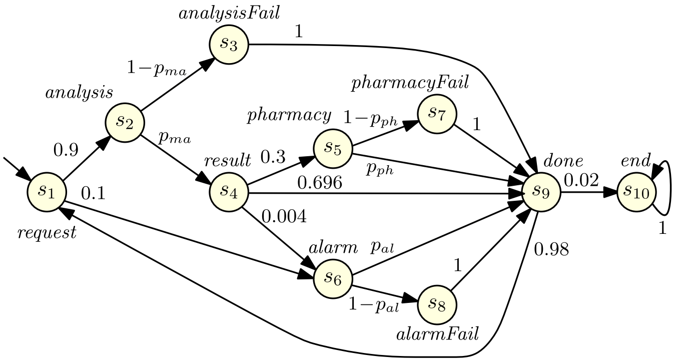


Fig. 1. Parametric Markov chain modelling the TAS workflow.
Source: Adapted from Calinescu et al. (2013).

We suppose that a team of software engineers wants to verify whether the third-party services they consider for the implementation of the TAS system satisfy – at 95% confidence level – the nonfunctional requirements from Table 1. We assume that the three services are yet to be tested, and therefore the success probabilities p_{ma}, p_{ph} and p_{al} for the medical analysis service, pharmacy service and alarm service, respectively, are unknown continuous variables, as specified in Definition 3. As such, the Markov chain that the engineers can use to verify the TAS requirements is parametric (Fig. 1), and the three services must be tested to observe how many of their executions succeed and how many fail (e.g., by not finishing in a timely manner). With these observations, the engineers can use formal verification with confidence intervals (cf. Section 2.3) to compute 95% confidence intervals for the probabilities from the three TAS requirements, which are formally expressed in PCTL in the last column from Table 1. Furthermore, once enough observations are available, these confidence intervals will be sufficiently narrow to ensure that the bounds from the requirements in Table 1 (i.e., 0.26 for requirement R1, 0.04 for R2, and 0.0003 for R3) fall outside the intervals, allowing the engineers to verify whether the requirements are satisfied or not.

However, under the realistic assumption that service invocations take non-negligible time, the engineers will want to complete this verification with as few invocations (i.e., observations) of each service as possible. Minimising this testing effort is particularly important when the verification needs to be performed at runtime, e.g., to find a suitable replacement for a failed component of a system, or when testing a system component has some other cost associated with it (e.g., an invocation charge paid to the provider of a service, or using battery energy on an embedded system). Deciding how many observations to obtain for each service in order to complete the verification of the requirements with minimal testing effort is very challenging. Our VERACITY verification approach addresses this challenge as described in the next section.

4. The VERACITY verification approach

4.1. Problem definition

The VERACITY verification approach is applicable to systems comprising $m > 1$ components that can be tested independently. We consider a component-based system whose $n \geq 1$ nonfunctional requirements are of the form

$$\mathit{prop}_i \bowtie_i \mathit{bound}_i, \tag{4}$$

where, for all $i \in \{1, 2, \ldots, n\}$, prop_i is a nonfunctional system property (e.g., reliability or, through the use of properties that compute the expected reward accumulated to reach states that model the completion of operations, response time), $\bowtie_i \in \{<, \leq, \geq, >\}$, $\mathit{bound}_i \in \mathbb{R}$ places a constraint on the acceptable values of prop_i, and the ith requirement can be expressed as a PCTL formula $\mathcal{P}_{\bowtie_i \mathit{bound}_i}[\cdot]$ or $\mathcal{R}_{\bowtie_i \mathit{bound}_i}[\cdot]$ over a parametric Markov chain $M = (S, s_0, \mathbf{P}, L)$. Given such a system, the verification problem addressed by VERACITY is to verify whether the n nonfunctional requirements (4) are satisfied at confidence level $\alpha \in (0, 1)$:

1. with minimum overall testing cost;
2. by using a (possibly empty) initial set of observations given by an observation function $O_0 : Z \times S \rightarrow \mathbb{N}$ with the semantics from (3);
3. by obtaining additional observations through unit-testing the m system components as required, where each unit test of the jth component: (i) generates one additional observation of an outgoing transition for every state in a non-empty set $Z_j \subset Z$, such that the state sets $Z_1, Z_2, \ldots, Z_m$ are disjoint and $\bigcup_{j=1}^{m} Z_j = Z$; and (ii) has an associated cost cost_j, that may represent testing time, resources, price, or a combination thereof.

Using the notation $[l_i, u_i]$ to denote the α-confidence interval that can be computed for prop_i from (4) after obtaining $n_1, n_2, \ldots, n_m$ additional observations for component $1, 2, \ldots, m$, respectively, the problem addressed by VERACITY is to find $n_1, n_2, \ldots, n_m \geq 0$ such that the overall testing cost

$$\sum_{j=1}^{m} n_j \mathit{cost}_j \text{ is minimised} \tag{5}$$

subject to

$$\begin{aligned} \forall i = 1..n : & (\bowtie_i \in \{<, \leq\} \wedge u_i \bowtie_i \mathit{bound}_i) \\ & \vee (\bowtie_i \in \{>, \geq\} \wedge l_i \bowtie_i \mathit{bound}_i) \end{aligned} \tag{6}$$

or

$$\begin{aligned} \exists i = 1..n : & (\bowtie_i \leq \wedge\ l_i \geq \mathit{bound}_i) \vee (\bowtie_i = \leq \wedge\ l_i > \mathit{bound}_i) \\ & \vee (\bowtie_i \geq \wedge\ u_i \leq \mathit{bound}_i) \vee (\bowtie_i = \geq \wedge\ u_i < \mathit{bound}_i). \end{aligned} \tag{7}$$

The constraints (6) and (7) correspond to the scenarios where all requirements (4) are satisfied and where at least one of requirements (4) is violated (meaning that the set of requirements as a whole is violated), respectively.

Due to the epistemic uncertainty associated with this verification problem and the stochastic nature of the component-testing results, a strategy guaranteed to achieve a minimum overall testing cost does not exist. We illustrate this limitation with an example. Consider a system that uses two web services, A and B. This system implements a simple workflow: first, it invokes service A, which is available with probability p_A; next, it invokes service B, which is available with probability p_B; next, it stops. Suppose that we want to establish whether the probability of successful invocation of both services is at least 0.9 (i.e., whether $p_A p_B \geq 0.9$) at confidence level $\alpha = 0.95$. If the two unknown probabilities are $p_A = 0$ (i.e., service A is never available) and $p_B = 0.95$, then allocating all the testing effort to unit-testing service A is the cheapest strategy for establishing that the requirement is violated, as this strategy will quickly show that p_A cannot be large enough for the requirement to be satisfied. Conversely, if $p_A = 0.95$ and $p_B = 0$, unit-testing only service B is the cheapest strategy. However, with no prior knowledge about p_A and p_B, it is impossible to always choose the best testing strategy. Therefore, our objective is to achieve an overall testing cost that is, on average, significantly lower than the cost associated with uniformly testing all components. Furthermore, for practical reasons, we add the constraint that the overall cost does not exceed a predefined testing budget $budget \in \mathbb{N}$.

Example 1. Consider the TAS system from our motivating example. Its $n = 3$ nonfunctional requirements R1–R3 from Table 1 are of the form in (4), are expressed as PCTL formulae over the parametric Markov chain from Fig. 1, and need to be verified at confidence level $\alpha = 0.95$. The set of Markov chain states with unknown outgoing transition probabilities is $Z = \{s_2, s_5, s_6\}$, and the (empty) initial set of observations is defined by the function $O_0(z, s) = 0$ for any $(z, s) \in Z \times S$. The system comprises $m = 3$ components that can be tested independently: the medical analysis service (component 1), the pharmacy service (component 2) and the alarm service (component 3). Additionally, invoking one of these services once provides an additional observation of an outgoing transition for the state in one of the disjoint sets $Z_1 = \{s_2\}$, $Z_2 = \{s_5\}$ and $Z_3 = \{s_6\}$, where $Z_1 \cup Z_2 \cup Z_3 = Z$. Finally, to fully cast the task of verifying requirements R1–R3 in the format from our problem definition, we assume that a testing budget $budget = 150\,000$ is available to complete the verification, and that the per-invocation costs of testing the three services are $cost_1 = 1$, $cost_2 = 1$ and $cost_3 = 2$, e.g., based on the ratios between their mean execution times. We chose a very large budget to ensure that the verification completes without exhausting the budget; this may be unacceptable in practice, in which case the verification might terminate without a decisive result (e.g., if the actual values of some of the n nonfunctional properties are extremely close to their associated bounds).

4.2. VERACITY verification process

To solve the problem from Section 4.1, VERACITY employs the iterative verification process depicted in Fig. 2. Each *round* (i.e., iteration) of this process comprises the four steps described below.

In the first step, formal verification with confidence intervals (Calinescu et al., 2015) is used to compute confidence intervals $[l_1, u_1]$, $[l_2, u_2]$, …, $[l_n, u_n]$ at confidence level α and (as a byproduct, as explained in Section 2.3) closed-form expressions $expr_1, expr_2, \ldots, expr_n$ for the n properties from the nonfunctional requirements (4). The observations O used to compute the n confidence intervals include the initial observations O_0 and, starting with the second iteration, all the additional observations O' obtained in the previous iterations of the process. If the observation set O_0 is empty, then the confidence interval $[l_i, u_i]$ computed for the ith property in the first iteration is $[0, 1]$ or $[0, \infty)$, depending on whether the ith requirement is of the form $\mathcal{P}_{\bowtie_i bound_i}[\,\cdot\,]$ or $\mathcal{R}_{\bowtie_i bound_i}[\,\cdot\,]$.

In the second step, VERACITY checks whether the n confidence intervals are sufficiently narrow for either constraint (6) or constraint (7) to hold. If either of these conditions are met, the verification problem is solved, and a verification result is produced. Otherwise, additional observations are needed to complete the verification. Two cases are possible in this situation. First, when the testing *budget* was fully utilised in the previous VERACITY rounds, the process terminates with an inconclusive 'budget exhausted' result. Otherwise, testing budget is still available, and the VERACITY adaptive uncertainty reduction heuristic detailed in Section 4.3 is used to calculate the numbers of additional component observations $nobs_1$, $nobs_2$, …, $nobs_m$ for the next round of the verification process, where

$$\sum_{j=1}^{m} nobs_j cost_j \approx rbudget \tag{8}$$

and $rbudget \in [0, budget]$ is a parameter of the VERACITY approach called the *round budget*.[5] The heuristic is adaptive in the sense that these numbers of additional observations vary from round to round, as the heuristic takes into account the actual observations from all the previous rounds (and any initial observations that may be available). The maximum number of rounds for the verification process is $\lceil budget/rbudget \rceil$. Accordingly, larger round budgets yield fewer rounds, and therefore less opportunity for adaptation but lower overheads (due to the fewer rounds). In contrast, smaller round budgets lead to more rounds, which offer more opportunity for adaptation but also come with higher overheads.

In the third step of the VERACITY process, $nobs_j$ tests of component j are carried out for $j = 1..m$. As we will explain in Section 5, these tests can be fully automated, or can be performed by a software engineer when requested by the VERACITY verification tool. The results of these tests are then encoded as an observation function O' with the format from (3).

Finally, in the fourth and last step of VERACITY, the new observations O' are integrated with all the observations obtained in the previous rounds of the process and the initial observations O_0, and the combined set of all available observations O is used in the next round of the verification process.

Example 2. For the three requirements for the TAS system from our motivating example, $\bowtie_1 = \bowtie_2 = \bowtie_3 \leq$. Accordingly, the requirements will be verified as satisfied at 95% confidence level if the observations acquired over successive rounds of the VERACITY verification process (and within the available *budget*) lead to the calculation of 95% confidence intervals $([l_i, u_i])_{i=1..3}$ that satisfy $u_1 < bound_1$, $u_2 < bound_2$ and $u_3 < bound_3$. If, on the other hand, $l_i \geq bound_i$ for any $i \in \{1, 2, 3\}$ in one of the verification rounds, the verification process will decide that requirement i is violated at 95% confidence level, and will terminate in that round. Finally, if the testing *budget* is used up before sufficient observations of the medical analysis, pharmacy and alarm services are obtained to allow either of these decisions to be made, the verification process will terminate with a 'budget exhausted' outcome.

[5] Equality cannot be always achieved in (8) because $nobs_1$, $nobs_2$, …, $nobs_m$ must take non-negative integer values.

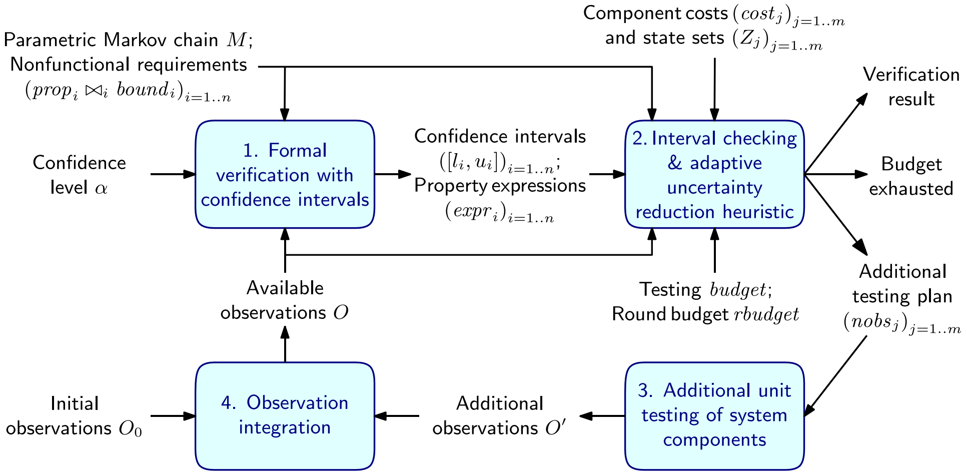

Component costs $(cost_j)_{j=1..m}$
and state sets $(Z_j)_{j=1..m}$
Parametric Markov chain M;
Nonfunctional requirements
$(prop_i \bowtie_i bound_i)_{i=1..n}$
Verification result
Confidence level α
1. Formal verification with confidence intervals
Confidence intervals $([l_i, u_i])_{i=1..n}$;
Property expressions $(expr_i)_{i=1..n}$
2. Interval checking & adaptive uncertainty reduction heuristic
Budget exhausted
Additional testing plan $(nobs_j)_{j=1..m}$
Available observations O
Testing $budget$;
Round budget $rbudget$
Initial observations O_0
4. Observation integration
Additional observations O'
3. Additional unit testing of system components

Fig. 2. Iterative four-step process of the VERACITY quantitative verification with adaptive uncertainty reduction.

4.3. Adaptive uncertainty reduction heuristic

4.3.1. Desiderata

Before describing the VERACITY heuristic for partitioning the round testing budget *rbudget* among the *m* system components, we present a set of desirable properties (i.e., *desiderata*) that we propose for any such heuristic:

D1. The requirements already verified as satisfied in previous rounds should not influence the partition of the round budget, as the uncertainty associated with these requirements has already been lowered enough to complete their verification.

D2. Reaching a resolution on requirements whose verification requires additional data (i.e., on requirements with $bound_i \in [l_i, u_i]$) that are *likely* to be violated should be prioritised when the round budget is partitioned. This desideratum captures the fact that verifying a requirement as violated ends the verification process immediately. Such requirements can be identified by noting that their $bound_i$ is very close to the "wrong" end of the confidence interval $[l_i, u_i]$. For instance, if $bound_i$ were much closer to l_i than to u_i, narrowing down the confidence interval $[l_i, u_i]$ even slightly has a good chance (but is, of course, not certain) of showing that requirement *i* is violated, and of terminating the verification process.

D3. If several requirements whose verification requires additional data influence the partition of *rbudget*, those requirements whose $bound_i$ value is closer to the middle of the confidence interval $[l_i, u_i]$ should have a bigger influence. This desideratum reflects the fact that the verification of such requirements is particularly affected by epistemic uncertainty, as a significant narrowing of their confidence intervals is likely to be needed in order to decide whether they are satisfied or violated.

D4. The *rbudget* fraction allocated to each component should reflect the sensitivity of the properties $prop_1$ to $prop_n$ from (4) to the parameters of that component. For instance, if the closed-form expression for the *i*th requirement is $expr_i = p_1 + 0.01p_2$, where p_1 and p_2 are probabilities associated with components C_1 and C_2, respectively, component C_1 should be allocated a larger *rbudget* fraction than C_2 (all other factors being equal).

D5. When the testing of different components is expected to yield similar uncertainty reductions, the round budget partition should prioritise the components with lower testing costs: the value of a new observation of a component is given by the ratio between the (expected) reduction in uncertainty brought by the observation and the cost of that observation.

4.3.2. Algorithm

The numbers of new component observations $nobs_1$, $nobs_2$, …, $nobs_m$ for each VERACITY verification round are computed by function NewObs from Algorithm 1. This function takes the following arguments (cf. Fig. 2):

- the round testing budget *rbudget*;
- the Markov chain *M* and the requirements $(prop_i \bowtie_i bound_i)_{i=1..n}$;
- the property confidence intervals $([l_i, u_i])_{i=1..n}$ and expressions $(expr_i)_{i=1..n}$ obtained in the previous step of the round;
- the component testing costs $(cost_j)_{j=1..m}$ and associated state sets with unknown transition probabilities $(Z_j)_{j=1..m}$;
- the observations *O* available at the start of the round;
- the configuration parameters $\epsilon_1, \epsilon_2 \in (0, 1)$, whose role is described later in this section.

The algorithm has three parts. In the first part (lines 2–7), it identifies the set of *relevant requirements R* that will influence the partitioning of the round budget. According to desideratum D1, the set *U* of requirements whose verification requires additional data is computed in line 2. Next, the if statement in lines 3–7 checks if $bound_i$ of any requirement from *U* is much closer (i.e., $1/\epsilon_1$ times closer) to the "wrong" end of the confidence interval $[l_i, u_i]$ than to the "right" end, where:

- the "wrong" end of $[l_i, u_i]$ is the end beyond which requirement *i* is violated, i.e., l_i if $\bowtie_i \in \{<, \leq\}$, and u_i otherwise;
- the helper functions Wrong, Right return the respective ends of $[l_i, u_i]$;
- $\epsilon_1 \in (0, 1)$ is a VERACITY configuration parameter.

As explained in desideratum D2, requirements with this property are likely to be violated. Therefore, if any such requirements exist, only the requirement most likely to be violated is retained in the relevant requirement set *R* (line 4). Otherwise, *R* is initialised to include all the requirements whose verification requires additional data (line 6).

The second part of the algorithm (lines 8–16) starts by using the observations *O* to calculate estimates for each unknown transition probability (i.e., parameter) associated with a state from $Z = \bigcup_{j=1}^{m} Z_j$ (line 8). This calculation is performed by the auxiliary function EstimateParams, which estimates the unknown transition probabilities between each state $z \in Z$ and each state $s \in$

Algorithm 1 Adaptive uncertainty reduction heuristic

```
1:  function NewObs($rbudget, M, (prop_i \bowtie_i bound_i)_{i=1..n}, ([l_i, u_i])_{i=1..n}, (expr_i)_{i=1..n}, (cost_j)_{j=1..m}, (Z_j)_{j=1..m}, O, \epsilon_1, \epsilon_2$)
2:      $U = \{i \in 1..n \mid bound_i \in [l_i, u_i]\}$                                         ▷ desideratum D1
3:      if $\exists i \in U : |bound_i - \text{Wrong}(\bowtie_i, l_i, u_i)|/|bound_i - \text{Right}(\bowtie_i, l_i, u_i)| < \epsilon_1$ then   ▷ desideratum D2
4:          $R \leftarrow \{\text{argmin}_{i \in U} |bound_i - \text{Wrong}(\bowtie_i, l_i, u_i)|/|bound_i - \text{Right}(\bowtie_i, l_i, u_i)|\}$
5:      else
6:          $R \leftarrow U$
7:      end if
8:      $paramEstimate \leftarrow \text{EstimateParams}(M, O)$
9:      $(relevance_j \leftarrow 0)_{j=1..m}$
10:     for $i \in R$ do
11:         $weight = (u_i - l_i)/\max\{|bound_i - (l_i + u_i)/2|, \epsilon_2\}$                 ▷ desideratum D3
12:         for $j = 1$ to $m$ do
13:             $sensitivity \leftarrow \sum_{p \in \text{Params}(M, Z_j)} \left|\frac{\partial expr_i(paramEstimate)}{\partial p}\right|$   ▷ desideratum D4
14:             $relevance_j \leftarrow relevance_j + weight \cdot sensitivity$
15:         end for
16:     end for
17:     for $j = 1$ to $m$ do
18:         $nobs_j \leftarrow \left\lfloor rbudget \cdot \frac{relevance_j}{\sum_{k=1}^{m} relevance_k} \cdot \frac{1}{cost_j} \right\rfloor$   ▷ desideratum D5
19:     end for
20:     return $(nobs)_{j=1..m}$
21: end function
```

S using the observed transition frequency $O(z,s)/\sum_{s' \in S} O(z,s')$. The special case when $\sum_{s' \in S} O(z,s') = 0$ for one or more states $z \in Z$ may be encountered in the first round, as we allow an empty initial set of observations O_0 (cf. Section 4.1). In this case, which we do not show in Algorithm 1 in order to keep the pseudocode simple, EstimateParams raises an exception and the round budget is split uniformly between the components whose state sets Z_j include states with zero observations.

Next in this part of the algorithm, a component relevance measure $relevance_j$ is first initialised in line 9, and then updated with contributions corresponding to the relevant requirements R by the for loop from lines 10–16. Each such contribution is the product of two factors (line 14) that correspond to desiderata D3 and D4, respectively:

- *weight*, a factor calculated as the ratio between the width of the confidence interval $[l_i, u_i]$ and the distance between $bound_i$ and the middle of the interval $[l_i, u_i]$ (line 11, where a small VERACITY configuration parameter $0 < \epsilon_2 \ll 1$ is used to prevent a division by zero);
- *sensitivity*, a measure of the sensitivity of expression $expr_i$ to the epistemic uncertainty affecting the parameters of component j, calculated by summing the absolute value of the partial derivatives of $expr_i$ with respect to every parameter of component j (line 13),[6] where the set of all such parameters is provided by the auxiliary function Params, and the partial derivatives are evaluated for the parameter values estimated in line 8.

The third and final part of the algorithm (lines 17–19) decides the number of new observations for each component based on the relevance and testing cost of that component. In accordance with desideratum D5, the number of new observations for component j is calculated (in line 18) by allocating to the component a fraction of $\frac{relevance_j}{\sum_{k=1}^{m} relevance_k}$ of *rbudget*, and dividing this "component budget" by $cost_j$.

To avoid ending with $nobs_1 = nobs_2 = \ldots = nobs_m = 0$, it is sufficient to use a round budget that satisfies the constraint $rbudget \geq \sum_{j=1}^{m} cost_j$ because:

$$
\begin{aligned}
&\forall j = 1..m : nobs_j = 0 \\
&\implies \forall j = 1..m : rbudget \cdot \frac{relevance_j}{\sum_{k=1}^{m} relevance_k} < cost_j \\
&\implies \sum_{j=1}^{m} \left(rbudget \cdot \frac{relevance_j}{\sum_{k=1}^{m} relevance_k} \right) < \sum_{j=1}^{m} cost_j \\
&\implies rbudget < \sum_{j=1}^{m} cost_j.
\end{aligned}
$$

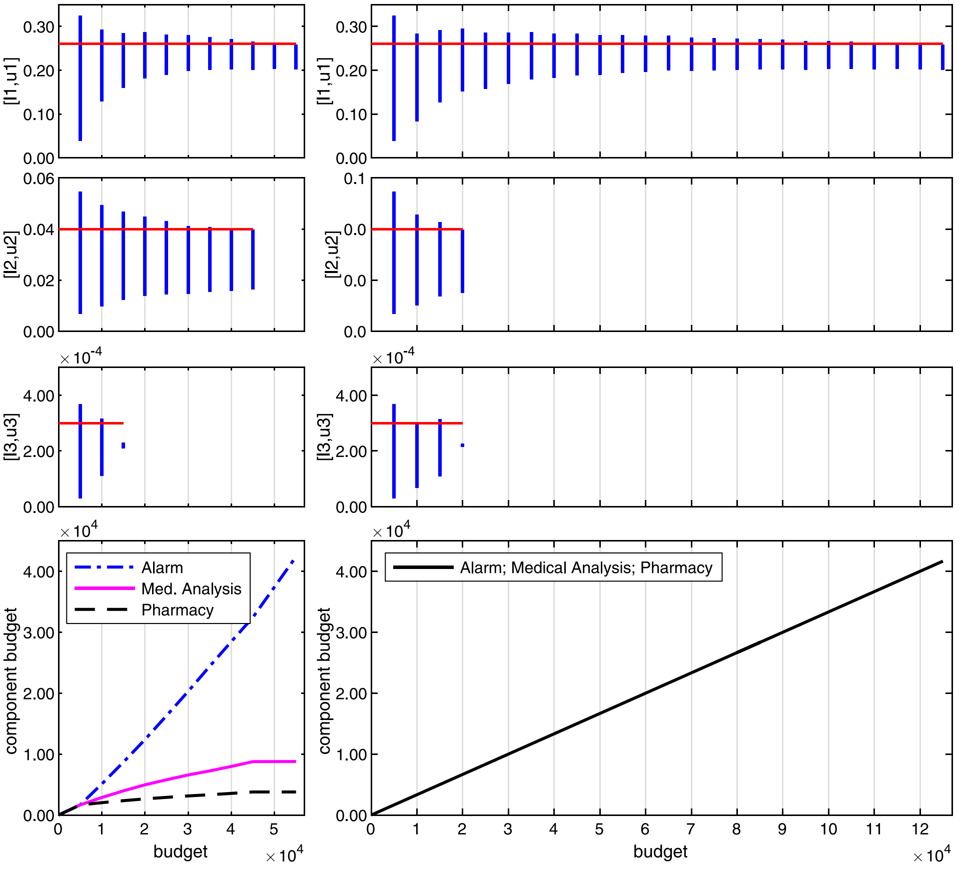
To achieve good progress with the verification process, *rbudget* should in fact be much larger (e.g., at least one order the magnitude larger) than $\sum_{j=1}^{m} cost_j$ in practice.

For improved readability, a couple of efficiency improvements are not included in Algorithm 1. In particular, the function Params and the partial derivatives required for factor *sensitivity* (line 13) can be precomputed once (in the first round of the VERACITY verification process), as the SUV parameters associated with a component do not change; only the evaluations of the precomputed partial derivatives need to be done in each round, for the new *paramEstimate* from that round.

With these improvements in place, the complexity of algorithm is $O(mn)$, because of the two nested for loops from lines 10–16 and 12–15, respectively. This is typically negligible compared to the complexity of the formal verification with confidence intervals and the additional unit testing from steps one and three of the VERACITY verification process, respectively.

Example 3. Fig. 3 shows the difference between the verification of the TAS nonfunctional requirements from Table 1 using the VERACITY verification process from Fig. 2 with: (a) our adaptive uncertainty reduction heuristic from Algorithm 1; and (b) our heuristic replaced with a uniform splitting of the round testing budget among the three system components. These results were obtained assuming that the unknown probabilities from the Markov chain in Fig. 1 were $p_{al} = 0.94$, $p_{ma} = 0.99$ and $p_{ph} = 0.95$, and using random number generators to synthesise additional observations O' based on these probabilities in the additional unit testing step of the VERACITY verification process from Fig. 2. The verification was performed with a round budget $rbudget = 5000$, unlimited overall testing *budget*, and the default

[6] Our *sensitivity* measure resembles the Birnbaum's measure of component importance for multicomponent systems (Birnbaum, 1969), which is often used in fault tree analysis (Barlow and Proschan, 1975).



(a) VERACITY

(b) uniform uncertainty reduction

Fig. 3. Verification of the nonfunctional requirements for the TAS system from the motivating example using (a) VERACITY adaptive vs. (b) uniform uncertainty reduction.

values $\epsilon_1 = 0.15$ and $\epsilon_2 = 10^{-6}$ for the two parameters of the VERACITY heuristic from Algorithm 1.

The top three pairs of graphs from Fig. 3 show the 95% confidence intervals $([l_i, u_i])_{i=1..3}$ (depicted as vertical lines) for the nonfunctional properties from the three TAS requirements from Table 1. These confidence intervals become narrower as additional observations are obtained in each round of the verification process, until they are narrow enough to fit completely under the $bound_i$ threshold (drawn as a horizontal line) from their associated requirement, i.e., until $u_i < bound_i$. As soon as this condition is met for a confidence interval $[l_i, u_i]$, that interval is no longer calculated in subsequent verification rounds. When the condition is met for all three confidence intervals, the epistemic uncertainty was reduced sufficiently to conclude that all requirements are satisfied, and the verification process terminates successfully. As shown by these graphs, VERACITY and the uniform uncertainty reduction method finish the verification of each requirement after a different number of verification rounds, and VERACITY completes the verification of the entire set of requirements with an overall testing cost of 55,000 compared to a 127% higher overall testing cost of 125,000 for the approach based on uniform uncertainty reduction. We also note in these graphs that the lower bounds of confidence intervals are not always increasing and the upper bounds of confidence intervals are not always decreasing from one VERACITY iteration to the next. As explained in Calinescu et al. (2015), this is because the calculation of confidence intervals for the multinomially distributed transition probabilities (and therefore also for the properties) of Markov chains is conservative, so increasing the number of observations may occasionally widen the confidence interval slightly. This does not affect the validity of formal verification with confidence intervals.

The bottom pair of graphs from Fig. 3 shows the cumulative testing cost per system component. When uniform uncertainty reduction is used, this cost is identical for all components, and is growing linearly with the number of verification rounds. In contrast, for the VERACITY approach, the cumulative cost grows at different rates for different components. Furthermore, the rate of growth for any single component varies across verification rounds because VERACITY continually *adapts* its partitioning of the round budget to the observations acquired in previous rounds, and to the effect that these observations have on confidence intervals $([l_i, u_i])_{i=1..3}$.

5. Implementation

We implemented the VERACITY verification process as a Java tool built on top of our FACT model checker (Calinescu et al., 2016). The VERACITY tool takes as input: a parametric Markov chain M expressed in the PRISM modelling language (Kwiatkowska et al., 2011) and annotated with the component costs $\left(cost_j\right)_{j=1..m}$ and state sets $\left(Z_j\right)_{j=1..m}$, and with the initial observations O_0; a set of PCTL-encoded nonfunctional requirements; and a confidence level α.

The overall testing *budget* and round testing budget *rbudget* are specified via a configuration file. In addition, this configuration file allows the user to optionally specify a component testing script that the tool can execute with the command

```
testing-script j nobsj
```

in order to obtain $nobs_j$ additional observations for component j automatically in the third step of the VERACITY verification process (cf. Fig. 2). If provided, this script needs to run $nobs_j$ unit-test against component j (e.g., by invoking the appropriate third-party service for the TAS system from our motivating example), and to return the $nobs_j$ observations from these tests as a list of numbers of transitions from the states in Z_j to other states of the Markov chain M. Alternatively (i.e., if the testing script is not provided), the tool asks the user to supply the required $nobs_j$ observations interactively at each round of the verification process.

Our VERACITY tool uses the model checkers FACT (Calinescu et al., 2016) and PRISM (Kwiatkowska et al., 2011), as well as MATLAB[7] to compute the confidence intervals and property expressions in the first step of its verification process. As such, the tool supports the same fragment of PCTL as FACT, i.e., non-nested PCTL properties $\mathcal{P}_{=?}[\,\cdot\,]$ and all types of PCTL reward properties (cf. Section 2.3). The tool is freely available from our project website https://www.cs.york.ac.uk/tasp/VERACITY, together with detailed instructions and all the models, requirement sets, and results from this paper.

6. Evaluation

We evaluated VERACITY by performing an extensive set of experiments aimed at answering the following research questions (RQs).

RQ1. Does VERACITY reduce the testing budget needed to verify a set of nonfunctional requirements compared to the baseline approach that partitions the testing budget of each verification round equally among SUV components?

RQ2. How effective is VERACITY at reducing the testing budget in scenarios where the SUV components have different testing costs?

RQ3. What effect does adjusting the round budget have on the VERACITY testing budget and verification time?

Answering research questions RQ1 and RQ2 required the comparison of experimental data from the use of VERACITY and of the baseline approach. We carried out this comparison using established empirical methods from software engineering (Kampenes et al., 2007; Kitchenham et al., 2017; Madeyski and Kitchenham, 2018; Madeyski, 2009) as follows. We started by applying the Shapiro–Wilk normality test, and established (as detailed in Sections 6.2 and 6.3) that our experimental data were not normally distributed. As such, we used three non-parametric methods to compare the two approaches:

1. We used the non-parametric Wilcoxon signed-rank test (as recommended, for instance, in Kampenes et al. (2007) and Madeyski (2009)) to assess whether VERACITY completes the verification with a smaller testing budget than the baseline approach.
2. We computed the *probability of superiority* for dependent-groups (Grissom and Kim, 2005) to evaluate the effect size associated with the use of VERACITY—this is a robust non-parametric measure of effect size recommended for software engineering experiments (Kitchenham et al., 2017); and we generated scatter plots for the visual inspection of the experimental data.
3. We calculated the difference in overall testing cost between VERACITY and the baseline approach, and we report the median value of these differences (where a negative value indicates a reduction in cost). Additionally, we plotted box plots (enabling the easy visual examination) of the percentage difference between the testing budgets required by the two approaches. The systematic review from Kampenes et al. (2007) confirms that the comparison of the median values and the use of box plots are non-parametric measures used frequently for the analysis of software engineering experiments.

In line with the established practice, we used the Shapiro–Wilk test and the three non-parametric methods with the following thresholds: a Shapiro–Wilk test p value below 0.05 as indicating non-normality; and a Wilcoxon signed-rank test p value below 0.05, a probability of superiority above 0.5 and a negative median difference to indicate that VERACITY requires a lower testing budget than the baseline approach.

To assess the generality of VERACITY, we report experimental results from two case studies in which VERACITY was applied to software systems from different domains. The first case study was based on the TAS system from our motivating example. In the second case study, we applied VERACITY to the verification of an online shopping web application. This system is introduced in Section 6.1, followed by descriptions of the experiments carried out to address the three research questions in Sections 6.2–6.4, and by a discussion of threats to validity in Section 6.5.

To enable the reproducibility of our results, we have made all the models, properties and data from our experiments available on the VERACITY project website at https://www.cs.york.ac.uk/tasp/VERACITY, which also presents a third case study that uses VERACITY for a model with more complex parametric probability distributions, and a fourth case study that applies VERACITY to a larger, 18-parameter model.

6.1. Online shopping web application

The system we used for the second case study is an online shopping application adapted from Filieri et al. (2012). We modelled the shopping process implemented by this application using a parametric Markov chain that comprises a combination of known and unknown transition probabilities. The known transition probabilities correspond to application components that have been in use for a long time, and for which the values of these probabilities can be determined from application logs. In contrast, the unknown transition probabilities correspond to new versions of several components that the online shopping company's developers have re-implemented and want to evaluate through *A/B testing*.

A/B testing (Fabijan et al., 2018; Kohavi and Longbotham, 2017; Siroker and Koomen, 2013) is a method for testing a new online-application feature, or a new implementation of an existing feature. Frequently used by leading companies like Amazon, Facebook, Google and Microsoft, the method involves splitting the users of a web application into two sets, such that one set of users is given access to a version of the application that includes the new feature (or the new implementation of a feature), while the other set continues to use the standard version of the application. In this way, A/B testing allows companies to evaluate new features and components, and to decide whether they should be included in their online applications or not. In our case study, we use A/B testing to motivate the scenario where a system has two different versions, such that one is well known in its behaviour and the other is still open to experimentation.

For our case study, we assume that the engineers want to verify whether the nonfunctional requirements from Table 2 would be satisfied if four application components were to be replaced with new variants of those components. Furthermore, we assume that in order to limit the disruption of the user experience that may occur if these requirements are in fact violated, the company

[7] http://www.mathworks.co.uk/products/matlab.

Table 2
Nonfunctional requirements for the online shopping application.

ID	Requirement	PCTL formula
R1	The probability that customers complete the shopping process successfully shall be above $bound_1$.	$\mathcal{P}_{>bound_1}$[F *success*]
R2	The probability that the authentication component fails shall be below $bound_2$.	$\mathcal{P}_{<bound_2}$[F *authFail*]
R3	The average number of uses of new components per shopping session[a] shall exceed $bound_3$.	$\mathcal{R}_{>bound_3}$[F *done*]

[a]This is a measure of how far the application users progress with the shopping process before giving up or encountering a component failure.

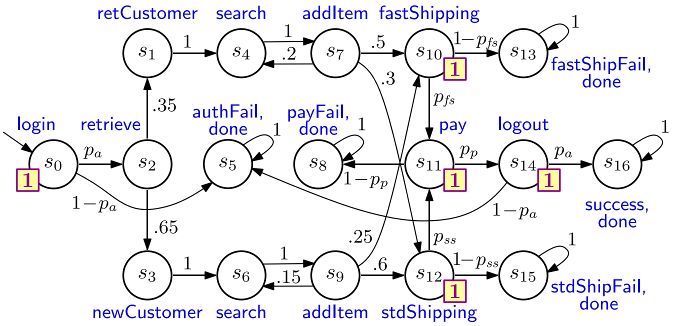


Fig. 4. Parametric Markov chain modelling the online shopping application. To enable the verification of requirement R3 from Table 2, the model is augmented with a reward structure that "counts" the uses of new components; this reward structure associates a reward $\rho(s_0) = \rho(s_{10}) = \rho(s_{11}) = \rho(s_{12}) = \rho(s_{14}) = 1$ with each state associated with the use of a new component (as shown in small squares next to these states) and zero rewards with the other states and with all state transitions of the Markov chain.

wants to perform this verification with as little A/B testing of each of the four new component implementations as possible.

The parametric Markov chain modelling the operation of the online shopping application is shown in Fig. 4. In the initial state (s_0) a customer attempts to login. We assume that the authentication web page is one of the components for which a new implementation needs to be tested, and therefore the probability that the customer can follow the authentication instructions and succeeds to login, denoted p_a, is unknown. If the authentication succeeds (state s_2), the customer is identified either as a returning customer (whose settings from the previous shopping session are restored, state s_1) or as a new customer (for whom default settings are used, state s_3). In both cases, the customer searches for items to purchase (states s_4/s_6) and adds them to the shopping basket (states s_7/s_9), until eventually all the required items are in the shopping basket and the customer moves to checkout where they select between two shipping options: fast shipping (state s_{10}) or standard shipping (state s_{12}). We assume that the probabilities of the incoming transitions into states s_1, s_3, s_4, s_6, s_7, s_9, s_{10} and s_{12} are known (from the previous use of the web application) and have the values from Fig. 4.

However, we assume that the web application components for selecting the two shipping options have been re-implemented, and therefore the probabilities p_{fs} and p_{ss} that the customer manages to use them successfully and to reach the payment state s_{11} are unknown. Likewise, we consider that a new version of the payment component has been implemented, and that the probability p_p that the customer manages to use it successfully (and to move to the logout state s_{14}) is unknown. Finally, we assume that the logout involves the use of the same new authentication component that was used for login, and therefore its (unknown) probability of succeeding is p_a.

6.2. Research question RQ1

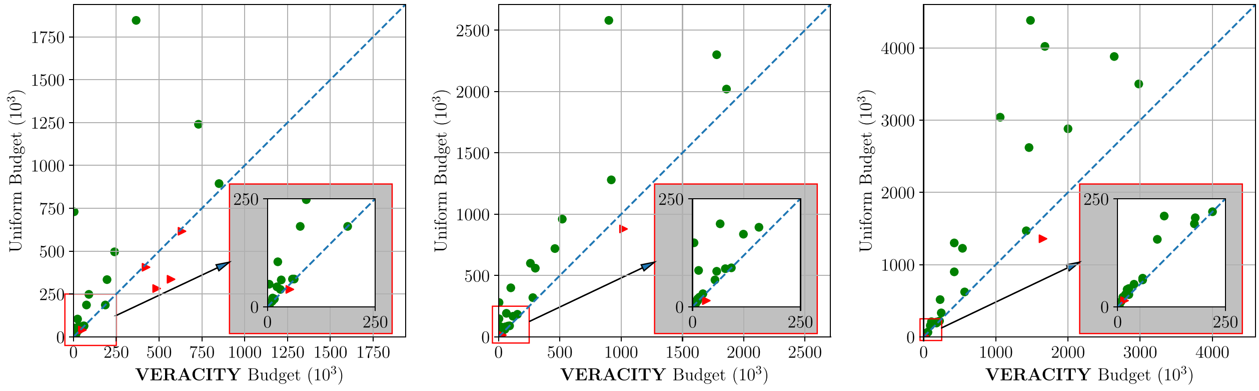
For each of the two systems used in our evaluation, we synthesised and examined a broad range of simulated scenarios (assuming unit component testing costs $cost_1 = cost_2 = \cdots = cost_m = 1$). In these scenarios, each parameter of the Markov chain M from Fig. 2 was given a deterministic value (drawn randomly from the interval [0, 1]) and test outcomes were sampled accordingly. Likewise, the bounds from the nonfunctional requirements (4) were given fixed values drawn randomly from the interval [0, 1]. We synthesised a sufficiently large number of scenarios to ensure that the impact of stochasticity in the experiments is low, and that the evaluation covered a combination of:

1. scenarios in which all requirements were satisfied: (i) by a narrow margin, i.e., the actual values of the properties from the nonfunctional requirements (4) were close to their associated bounds; (ii) by a wide margin; and (iii) some by a narrow margin and the others by a wide margin;
2. scenarios in which some of the requirements were satisfied and the remaining requirements were violated by: (i) a narrow margin; (ii) a wide margin; and (iii) some by a narrow margin and the others by a wide margin;
3. scenarios in which all requirements were violated by: (i) a narrow margin; (ii) a wide margin; and (iii) some by a narrow margin and the others by a wide margin.

In all the experiments, we used the VERACITY tool in conjunction with a simulated component-testing script with the characteristics described in Section 5. This script emulated the outcome of unit testing the SUV components by using a separate Java pseudorandom number generator for each component. To avoid any bias in the comparison of VERACITY with the baseline approach mentioned in research question RQ1, we used the same pseudorandom number generator seeds in the corresponding experiments for the two approaches.

Case study 1 (TAS). For the TAS system, we carried out experiments that examined the effectiveness of VERACITY for a set of 33 scenarios that were randomly generated as described at the beginning of this section. For each of these scenarios, the verification of the TAS nonfunctional requirements was carried out at three confidence levels: $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$. Finally, to answer research question RQ1, two experiments were performed for each scenario and each confidence level α: one in which we used the VERACITY uncertainty reduction heuristic, and one in which we used the baseline approach that partitions the testing budget of each verification round equally among the TAS components. In total, we performed 198 verification experiments, corresponding to 33 scenarios ×3 confidence levels ×2 uncertainty reduction methods.

The Shapiro–Wilk normality test showed that the 33 experimental data points are not normally distributed for any of the six combinations of uncertainty reduction method and confidence level ($p = 1.73 \times 10^{-7}$, $p = 1.91 \times 10^{-7}$ and $p = 2.31 \times 10^{-6}$ for VERACITY with $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$, respectively; and $p = 5.09 \times 10^{-6}$, $p = 4.89\times\ 10^{-8}$ and $p = 3.03 \times 10^{-6}$ for the baseline method with $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$, respectively). Therefore, we compared the VERACITY and baseline outcomes by using the three non-parametric methods mentioned at the beginning of Section 6, obtaining the following results, all of which confirm the superiority of VERACITY over the baseline approach:



(a) $\alpha = 0.90$ (b) $\alpha = 0.95$ (c) $\alpha = 0.99$

Fig. 5. [RQ1:TAS] Testing budgets required to complete the verification of the TAS nonfunctional requirements using the VERACITY and the uniform methods for partitioning the testing round budget among the components of the TAS system. The wide range of budgets required to complete the verification process for different scenarios reflects the variety of these scenarios: in some scenarios, the TAS requirements are satisfied or violated by a wide margin (so less testing is needed, as shown by the inset diagrams), whereas in others some or all of the requirements are satisfied or violated by a narrow margin (so much more testing is needed).

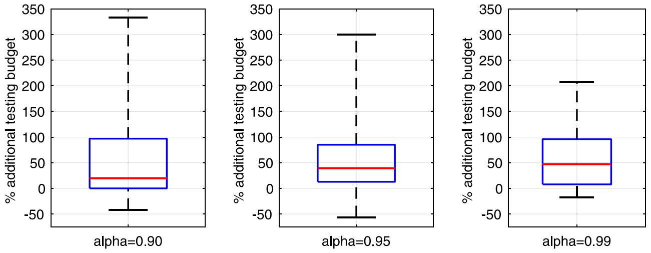


Fig. 6. [RQ1:TAS] Additional testing budget required to complete the verification of the TAS nonfunctional requirements when the round budget is partitioned using the uniform method instead of the VERACITY method. To ensure readability, the upper part of the boxplots is truncated, meaning that the outliers at 404%, 1200% and 18 150% (for $\alpha = 0.90$), and at 4252% and 6900% (for $\alpha = 0.95$) are not shown. No outliers exist below the bottom whisker of any of the boxplots.

1. for $\alpha = 0.90$, Wilcoxon $p = 0.003$, probability of superiority 0.697, and median difference -1988;
2. for $\alpha = 0.95$, Wilcoxon $p = 0.000$, probability of superiority 0.788, and median difference $-25\,957$;
3. for $\alpha = 0.99$, Wilcoxon $p = 0.000$, probability of superiority 0.848, and median difference $-25\,902$.

As mentioned at the beginning of Section 6, we also provide scatter plots and box plots enabling the visual inspection of the experimental data (Figs. 5 and 6).

These results show that VERACITY outperforms the baseline method by completing the verification process with smaller testing budgets for a great majority of the scenarios and at all confidence levels. In a few scenarios, the baseline method performs better than VERACITY (typically only marginally better). This is expected given the stochastic nature of the verified system, and the fact that the verification starts with no knowledge about the behaviour of the three TAS components. Finally, in a small number of additional scenarios, VERACITY achieves only modest testing cost savings. This is also expected, as the best way to reduce epistemic uncertainty in some verification scenarios is to partition the round testing budget approximately equally among the tested system components, and our approach manages to do this well.

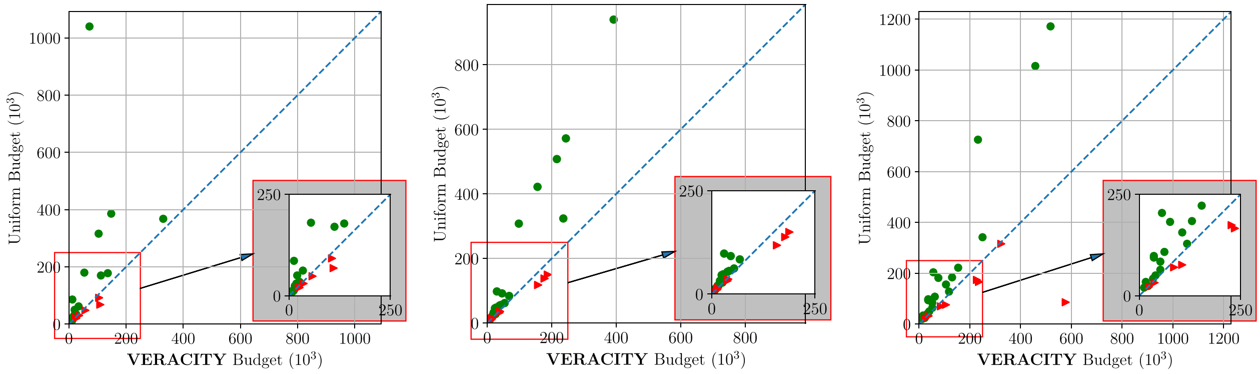
The experimental results show that the testing budget reductions enabled by VERACITY are particularly significant when the verification is carried out at higher confidence levels. This is extremely useful for two reasons. First, in real-world scenarios, the nonfunctional requirements of software systems should be verified with high levels of confidence ($\alpha = 0.99$ or even higher, as in e.g. medicine); deploying a system whose requirements were only verified at a low confidence level introduces significant risks. Second, the testing budget needed to complete the verification increases with the confidence level α, as the epistemic uncertainty needs to be reduced much more in order to make decisions with higher confidence. This increase of the required testing budget for larger α values is clearly visible in the scales of the graphs from Fig. 5. As such, the scenarios in which VERACITY reduces the cost of testing the most are: (i) of particular practical importance; and (ii) characterised by high testing costs, so the cost reductions achieved by our uncertainty reduction method are especially beneficial.

Case study 2 (online shopping web application). To assess the effectiveness of VERACITY for the online shopping web application (WebApp), we performed a similar suite of experiments to those described for the TAS case study. This time, we examined the ability of VERACITY to reduce testing costs compared to the baseline uncertainty reduction method for the verification of 30 randomly generated scenarios. In each of the 30 scenarios, the verification of the WebApp requirements was carried out at three confidence levels ($\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$), for both the VERACITY and the uniform uncertainty reduction methods, giving a total of 180 experiments.

The 30 experimental data points are not normally distributed for any combination of uncertainty reduction method and confidence level (Shapiro–Wilk test $p = 6.35 \times 10^{-7}$, $p = 9.30 \times 10^{-6}$, and $p = 1.44 \times 10^{-6}$ for VERACITY with $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$, respectively; and $p = 1.30 \times 10^{-8}$, $p = 3.41 \times 10^{-7}$, and $p = 2.33 \times 10^{-7}$ for the baseline method with $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$, respectively). The Wilcox signed-rank test, the probability of superiority measure, and the median difference all indicate that VERACITY outperforms the baseline method for every confidence level:

- for $\alpha = 0.90$, Wilcoxon $p = 0.001$, probability of superiority 0.800, and median difference -5000;
- for $\alpha = 0.95$, Wilcoxon $p = 0.002$, probability of superiority 0.733, and median difference $-10\,000$;
- for $\alpha = 0.99$, Wilcoxon $p = 0.018$, probability of superiority 0.700, and median difference $-11\,000$;

Figs. 7 and 8 summarise the results of these experiments. As for the TAS system, VERACITY successfully reduces the testing budget required to complete the verification of the nonfunctional



(a) $\alpha = 0.90$ (b) $\alpha = 0.95$ (c) $\alpha = 0.99$

Fig. 7. [RQ1:WebApp] Testing budgets required to complete the verification of the WebApp nonfunctional requirements using the VERACITY and the uniform methods for partitioning the testing round budget among the components of the online shopping system.

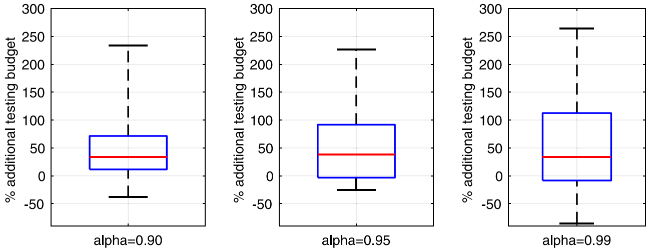


Fig. 8. [RQ1:WebApp] Additional testing budget required to complete the verification of the WebApp nonfunctional requirements when the round budget is partitioned using the uniform method instead of the VERACITY method. To ensure readability, the upper part of the boxplots is truncated at 300%, meaning that the outliers at 616% and 1344% (for $\alpha = 0.90$) are not shown. No outliers exist below the bottom whisker of any of the boxplots.

requirements, across a wide range of testing budget needs (where small testing budgets are needed when the requirements are satisfied or violated by a wide margin, and large budgets when some or all of the requirements are narrowly satisfied/violated). In the small number of scenarios where the uniform round budget partitioning method achieves better results, the overall testing budget is small, and the VERACITY-based verification is typically only marginally more expensive. Again, many significant budget reductions occur when (i) the baseline method budget is high and (ii) the requirements are verified at higher confidence levels. For instance, all of the baseline method budgets above 400,000 from Fig. 7 (one for $\alpha = 0.90$, four for $\alpha = 0.95$, and three for $\alpha = 0.99$) are at least halved by VERACITY.

Discussion. VERACITY is a heuristic whose behaviour depends on the configuration parameters ϵ_1 and ϵ_2 from Algorithm 1, on the configuration of the formal verification with confidence intervals step from Fig. 2, and on the stochasticity of its component testing outcomes. As such, it is expected that VERACITY cannot always outperform the baseline method, just like a superior medical treatment is unfortunately not always outperforming an inferior treatment. Therefore, we used established analyses methods from the software engineering domain to evaluate VERACITY, and the results of these analyses show its superiority over the baseline method. Additionally, our examination of scenarios in which VERACITY used larger testing budgets than the baseline method revealed that these were typically scenarios in which:

- the uniform budget partitioning used by the baseline method was the (nearly) optimal option, so the best VERACITY could have achieved was to match the performance of the baseline method—but this would have required perfectly chosen values for the VERACITY configuration parameters;
- the stochastic effects of the component testing required for the verification had an adverse effect on VERACITY;
- calibrating the values of the VERACITY configuration parameters (instead of using their default values) can improve the performance of our method;
- a combination of the previous three factors was at play.

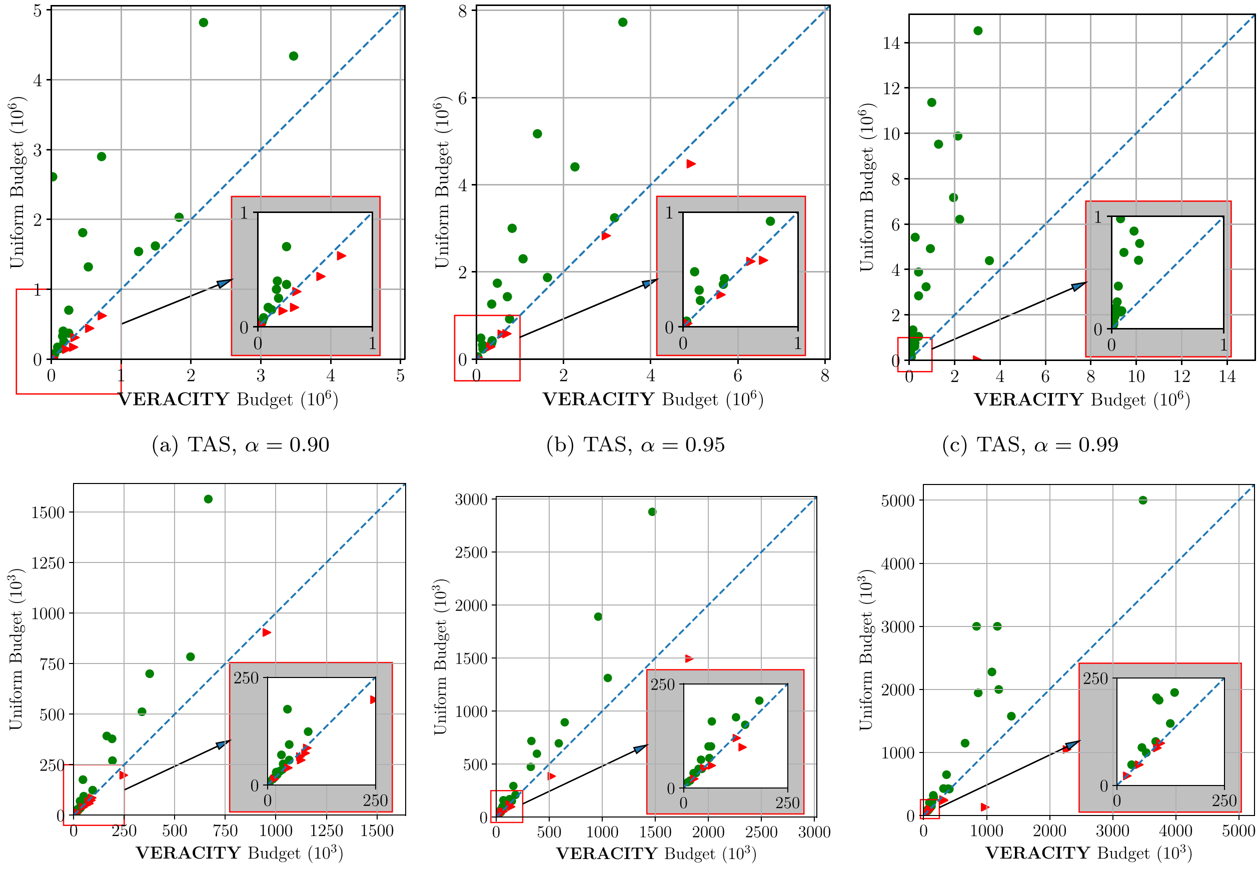
The influence of such factors is unavoidable for software engineering methods that employ heuristics, and – given the positive results of the analyses presented earlier in this section and in the rest of our evaluation – it does not affect the usefulness of VERACITY.

6.3. Research question RQ2

Experiments similar to those from Section 6.2 (but focusing on varying the component testing costs) were carried out to evaluate the ability of VERACITY to handle the practical situations where the costs of testing different components of a system are different. One situation when this is likely to be true is, for instance, when these costs represent the times required for the regression testing of a software system with several modified components (Muccini et al., 2006). Another situation when this may apply is for testing different web services at runtime, when a limited overall time (i.e., testing budget) is available to verify whether using these services as part of a service-based system like TAS satisfies a set of nonfunctional requirements (Calinescu et al., 2011). This is also likely to be true for the A/B testing of new features of an online application (Fabijan et al., 2018; Kohavi and Longbotham, 2017; Siroker and Koomen, 2013) like the shopping application from Section 6.1, where these costs may represent the different (expected) business impact of each of the new features not working as intended.

To evaluate the usefulness of VERACITY in such situations, we repeated all the experiments from Section 6.2 assuming different testing costs for the components of the TAS and WebApp systems from our two case studies. To this end, we took each of the 33 TAS verification scenarios and of the 30 WebApp verification scenarios, and we assigned randomly generated testing costs in the interval [1, 5] to the three TAS components and the four WebApp components, respectively.

The Shapiro–Wilk test indicated that the experimental results were not normally distributed: $p = 1.15 \times 10^{-7}$, $p = 7.64 \times 10^{-7}$, and $p = 6.58 \times 10^{-7}$ for VERACITY applied to TAS at $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$; $p = 7.35 \times 10^{-7}$, $p = 1.23 \times 10^{-6}$, and $p = 1.99 \times 10^{-6}$ for VERACITY applied to WebApp at $\alpha = 0.90$,



(a) TAS, $\alpha = 0.90$ (b) TAS, $\alpha = 0.95$ (c) TAS, $\alpha = 0.99$

(d) WebApp, $\alpha = 0.90$ (e) WebApp, $\alpha = 0.95$ (f) WebApp, $\alpha = 0.99$

Fig. 9. [RQ2] Testing budgets required to complete the verification process using the VERACITY and the uniform methods for partitioning the testing round budget among the components of the TAS and WebApp systems.

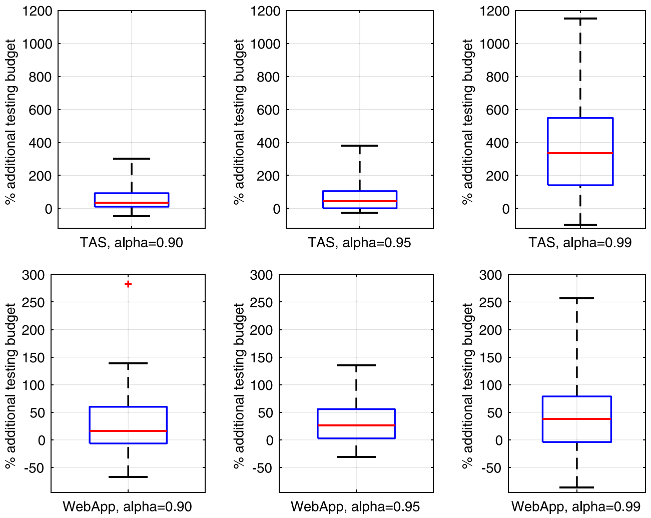


Fig. 10. [RQ2] Additional testing budget required to complete the verification of the TAS and WebApp nonfunctional requirements when the round budget is partitioned using the uniform method instead of the VERACITY method. To ensure readability, the upper part of the TAS boxplots is truncated at 1200%, meaning that two TAS outliers at 12 950% (for $\alpha = 0.90$) and at 1984% (for $\alpha = 0.99$) are not shown. No other hidden outliers exist for any of the boxplots.

$\alpha = 0.95$ and $\alpha = 0.99$; $p = 5.51 \times 10^{-7}$, $p = 2.97 \times 10^{-6}$ and $p = 3.98 \times 10^{-6}$ for the baseline method applied to TAS at $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$; and $p = 6.23 \times 10^{-7}$, $p = 6.06 \times 10^{-7}$ and $p = 1.87 \times 10^{-6}$ for the baseline method applied to WebApp at $\alpha = 0.90$, $\alpha = 0.95$ and $\alpha = 0.99$. We therefore applied the three non-parametric analysis methods, obtaining the following results for the TAS experiments:

- for $\alpha = 0.90$, Wilcoxon $p = 0.001$, probability of superiority 0.818, and median difference −29 973;
- for $\alpha = 0.95$, Wilcoxon $p = 0.002$, probability of superiority 0.758, and median difference −19 985;
- for $\alpha = 0.99$, Wilcoxon $p = 0.000$, probability of superiority 0.970, and median difference −639 617,

and the results below for the WebApp experiments:

- for $\alpha = 0.90$, Wilcoxon $p = 0.024$, probability of superiority 0.700, and median difference −4000;
- for $\alpha = 0.95$, Wilcoxon $p = 0.002$, probability of superiority 0.767, and median difference −19 000;
- for $\alpha = 0.99$, Wilcoxon $p = 0.001$, probability of superiority 0.733, and median difference −59 000.

These results indicate the superiority of VERACITY over the baseline approach, at all confidence levels and for both TAS and WebApp.

The testing budgets required to complete the verification process using the VERACITY and the baseline round-budget partitioning methods in these scenarios with different component testing costs are compared in Figs. 9 and 10. As in the scenarios with the same testing costs for all components, our VERACITY verification approach outperforms the baseline verification approach in the majority of the examined scenarios, often by a large margin.

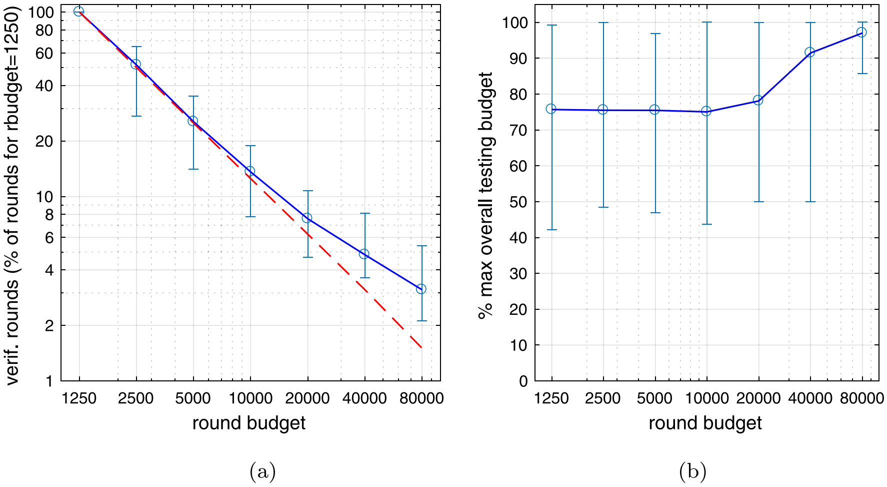


Fig. 11. [RQ3] Effect of varying the VERACITY round budget on (a) the number of verification rounds; and (b) a normalised measure of the overall testing budget (see main text for details). The plots show mean values and ranges over 10 randomly selected verification scenarios with unit component testing costs $cost_1 = cost_2 = \cdots = cost_m = 1$.

As shown in Fig. 10, this margin increases as α increases. This increase is particularly significant for the TAS system, where the median additional testing budget required by the baseline verification approach grows from 33% at $\alpha = 0.90$ to 42% at $\alpha = 0.95$, and 335% at $\alpha = 0.99$. This growth is less pronounced but still present for the WebApp system, where the median additional testing budget increases from 16% at $\alpha = 0.90$ to 26% at $\alpha = 0.95$, and 38% at $\alpha = 0.99$.

In the small number of scenarios where the baseline approach completes the verification within a smaller testing budget, the difference between this approach and VERACITY is typically modest, and/or occurs for scenarios where both approaches perform the verification with relatively small overall testing budgets.

6.4. Research question RQ3

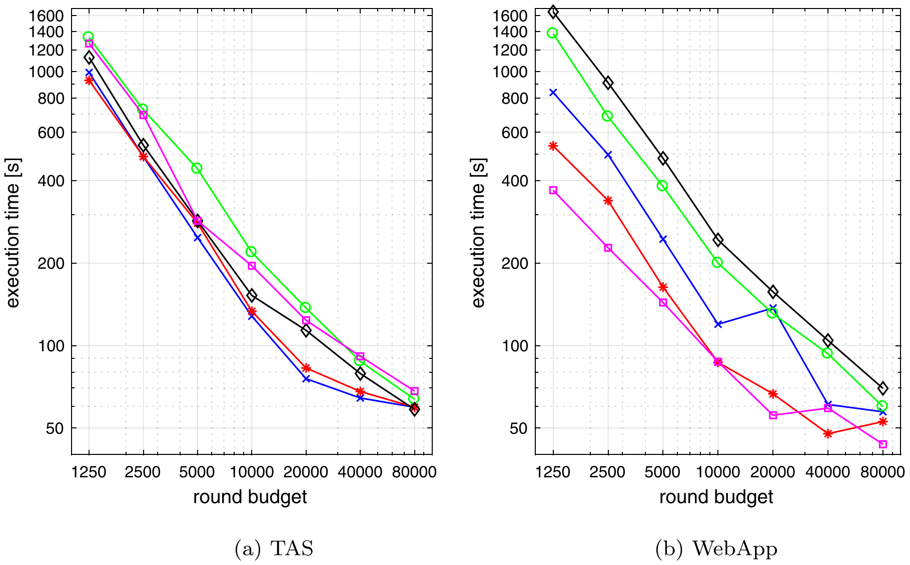
The round testing budget *rbudget* is a key parameter of VERACITY. Large *rbudget* values are undesirable because they lead to all the component observations needed to complete the verification of the nonfunctional requirements being acquired in a small number of verification rounds. This gives VERACITY limited opportunity to meaningfully adapt its partitioning of the round budget to the system and requirements being verified. Small *rbudget* values are equally undesirable because they yield only a few additional observations in each round. As such, they provide insufficient information to properly guide the round-budget partitioning in the early verification rounds, and lead to large numbers of verification rounds, which can be computationally expensive because of the formal verification with confidence intervals step of VERACITY.

To analyse these effects of *rbudget*, we randomly selected five of the TAS verification scenarios and five of the WebApp verification scenarios from Section 6.2, and we used VERACITY to verify the nonfunctional requirements of the two systems for each round budget value in the set $RB = \{1250, 2500, 5000, 10\,000, 20\,000, 40\,000, 80\,000\}$. The experimental results are presented in Fig. 11. First, the graph from Fig. 11(a) shows the number of verification rounds required when the verification was carried out using each of the round budgets from *RB*. The dashed line from this graph shows what the "ideal" effect of increasing the round budget would look like, i.e., a halving of the number of verification rounds each time when *rbudget* is doubled, from the baseline of 100% for $rbudget = 1250$ to 50% of that baseline for $rbudget = 2500$, 25% for $rbudget = 5000$, etc. In reality, the number of verification rounds is increasingly above the ideal value as *rbudget* grows, until it is above this ideal value for all 10 verification scenarios both for $rbudget = 40\,000$ and for $rbudget = 80\,000$. This indicates that very large *rbudget* values increase the overall testing budgets required by VERACITY—a finding that is further confirmed by Fig. 11(b), which shows how the overall testing budget necessary to complete the verification grows with *rbudget*.

To summarise the very different overall testing budgets required for our 10 randomly selected verification scenarios in a consistent way, Fig. 11(b) considers the overall testing budgets b_1, $b_2, \ldots, b_7$ associated with each verification scenario and the seven *rbudget* values from *RB*, finds $b_{\max} = \max\{b_1, b_2, \ldots, b_7\}$, and computes the percentages of $b_{\max}$ that $b_1, b_2, \ldots, b_7$ correspond to, i.e., $pb_1 = 100b_1/b_{\max}$, $pb_2 = 100b_2/b_{\max}$, ..., $pb_7 = 100b_7/b_{\max}$. These "normalised" budgets show the round budget for which VERACITY requires the highest overall testing budget (e.g., $pb_7 = 100$ means that the highest overall testing budget is needed when $rbudget = 80\,000$), and how the testing budgets for other *rbudget* values compare to that (e.g., $pb_1 = 75$ means that the overall testing budget for $rbudget = 1250$ is 75% of the highest overall testing budget). Fig. 11(b) shows how the mean of these normalised budgets increases from $pb_1 = 75.5\%$ for $rbudget = 1250$ to $pb_7 = 97\%$ for $rbudget = 80\,000$. The variability of the budget values is very large across the 10 verification scenarios from our experiments, except for the largest round budget $rbudget = 80\,000$, which indicates that this round budget is consistently too large across the majority of the scenarios.

While the effects of using very large *rbuget* values are clear in Fig. 11, noticing the effects of small *rbuget* values requires a more careful analysis of the experimental results. A first observation we can make is that the experiments with the smallest *rbudget* values of 1250, 2500 and 5000 used the largest number of verification rounds (as expected, see Fig. 11a) without delivering smaller mean overall testing budgets than the experiments for $rbudget = 10\,000$ (see Fig. 11b). In fact, the numerical results show a very slight decrease in the mean overall testing budgets from 75.7% for $rbudget = 1250$ to 75.5% for $rbudget = 2500$, 75.47% for $rbudget = 5000$ and 75.05% for $rbudget = 10\,000$. Thus, Fig. 11b illustrates the first undesirable effect of using small *rbudget* values, namely an increase in the cost of the component testing without enabling VERACITY to adapt its partition of the round budget more effectively.

The second undesirable effect of using small *rbudget* values is visible in Fig. 12, which depicts the end-to-end verification times



(a) TAS

(b) WebApp

Fig. 12. [RQ3] Effect of varying the round budget on the VERACITY execution time (experiments carried out on a c5.2xlarge Windows Server 2019 Amazon EC2 instance with 3.00 GHz Intel(R) Xeon(R) Platinum 8124M CPU, and 16 GB of memory assuming unit component testing costs $cost_1 = cost_2 = \cdots = cost_m = 1$).

for each of the five TAS verification scenarios and each of the five WebApp verification scenarios we used for the experiments described in this section. As shown by the logarithmic-scale graphs from this figure, the VERACITY execution times approximately double each time the round budget is halved from *rbudget* = 10 000 to *rbudget* = 5000, to *rbudget* = 2500 and, finally, to *rbudget* = 1250. Even when VERACITY is used at design time and execution times of close to 30 min (for *rbudget* = 1250) are acceptable, the results from Fig. 11b show that such long execution times yield no benefit, so very small *rbudget* are not recommended. As a side comment, we note that the VERACITY execution time does not always decrease monotonically in Fig. 12b. This is due to variations in the computation time required to calculate the confidence intervals in step 1 of the VERACITY verification process (cf. Fig. 2), to the stochasticity of the observations from the component testing performed in step 3 of this process, etc.

6.5. Threats to validity

Construct validity threats may arise due to assumptions made when modelling the systems from our case studies. To mitigate these threats, we used models and nonfunctional requirements based on established case studies from the research literature (Calinescu et al., 2011; Baresi et al., 2007; Filieri et al., 2012).

Internal validity threats may be caused by bias in establishing cause–effect relationships in our experiments. To limit these threats, we assessed VERACITY for large numbers of verification scenarios with randomly generated nonfunctional requirements bounds for each research question and each case study: 378 verification scenarios for research question RQ1, 378 verification scenarios for RQ2, and 70 verification scenarios for RQ3. Furthermore, as explained at the beginning of Section 6.2, we ensured that these experiments included scenarios where the requirements were satisfied, where they were violated, and where some requirements were satisfied and others were violated—both by a wide margin and by a narrow margin. Finally, we enable replication by making all experimental results available on our project's website.

External validity threats may exist if the verification of the nonfunctional requirements of other software systems cannot be expressed in the format from our problem definition in Section 4.1. We limited these threats by ensuring that VERACITY supports the verification of systems whose behaviour is modelled using parametric Markov chains encoded in the widely used modelling language of the PRISM model checker (Kwiatkowska et al., 2011), with nonfunctional requirements specified in the established temporal logic PCTL (Andova et al., 2004; Ciesinski and Größer, 2004; Hansson and Jonsson, 1994). Parametric Markov models are increasingly used to model software systems including service-based systems (Gerasimou et al., 2018), software product lines (Ghezzi and Sharifloo, 2013), software controllers of cyber–physical systems (Calinescu et al., 2017b), and multi-tier software architectures (Calinescu et al., 2021). Nevertheless, additional experiments are needed to establish the applicability and feasibility of VERACITY in domains with characteristics different from those used in our evaluation.

Another external validity threat may arise if the verification of other software systems requires the use of larger Markov models than those that we used to evaluate VERACITY. To mitigate this threat, we used systems and models proposed by other projects in established software engineering venues (Baresi et al., 2007; Filieri et al., 2012). Additionally, we assessed the level of this threat by comparing the size of these models to that of the discrete-time Markov chains (whether parametric or not) from all the research papers:

- published within the past five full years (2016–2020) in the *Journal of Systems and Software* and in all the CORE2020[8] rank A* software engineering journals (i.e., *IEEE Transactions on Software Engineering* and *ACM Transactions on Software Engineering and Methodology*) and conferences (i.e., the *International Conference on Software Engineering* and the *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*);
- that mention the size of the DMTCs used for evaluation, or provide these DTMCs or access to them on a project website or in a project repository.

Across the eight papers that met these criteria (Calinescu et al., 2017b; Su et al., 2016b; Franco et al., 2016; Llerena et al., 2018; Wang et al., 2018; Nakagawa et al., 2019; Afzal et al., 2019; Su

[8] https://www.core.edu.au/conference-portal.

et al., 2016a), models not larger than our TAS model (i.e., 10 states) are used for evaluation in five papers, and models smaller than or of the same size as our shopping application model (i.e., 17 states) are used for evaluation in all eight papers. While five papers also use models larger than ours for evaluation, these are not models of software systems with multiple components (to which VERACITY is applicable) but models of algorithms, programs and logical gates, or synthetic models that do not correspond to an actual software system. This analysis (whose detailed results are provided on our project's website) shows that abstraction allows many software engineering projects to usefully exploit models of similar size to the models used for evaluation in our paper. Nevertheless, larger models are sometimes required, and therefore we carried out an additional case study in which we applied VERACITY to an 18-parameter, 41-state discrete-time Markov chain modelling a larger component-based software system taken from Calinescu et al. (2021). This case study, reported on our project website https://www.cs.york.ac.uk/tasp/VERACITY, indicates that VERACITY can also handle models with larger numbers of parameters and states than those presented earlier in this section.

7. Related work

Within the past decade, the study of uncertainty in the modelling, analysis and verification of complex systems has attracted significant attention from the research community. As such, the existence of different classes of uncertainty is now widely recognised (Ramirez et al., 2012; Giese et al., 2014; Perez-Palacin and Mirandola, 2014; Walker et al., 2003), and the research literature provides multiple definitions of uncertainty. Most of these definitions classify uncertainties depending on their: (i) *level* (ranging from determinism to complete ignorance); (ii) nature (aleatory or epistemic); and (iii) source (in the structure or parameters of models, associated with changes in the operational environment or with the dynamics in the availability of resources, or due to changes in the user goals) (Walker et al., 2003; Garlan, 2010; Esfahani and Malek, 2013; Perez-Palacin and Mirandola, 2014).

When dealing with the verification of performance, reliability, or other nonfunctional requirements, the term uncertainty is often defined in terms of aleatory and epistemic uncertainty. The aleatory variability of parameters and indices is typically captured using stochastic modelling notations, while the epistemic uncertainty, which refers to the behaviour of system portions that are intrinsically unknown, requires ad-hoc methods.

The common goal of these methods is to introduce analysis methodologies able to produce satisfactory results even in presence of such types of uncertainty. For example, Trubiani et al. (2013), Meedeniya et al. (2011) and Meedeniya et al. (2012) propose methods that can be used at design time, to identify software architectures, configurations or component compositions that satisfy a given set of nonfunctional requirements. These approaches take into account the epistemic uncertainty associated with the parameters of models, and use probability distributions to model this uncertainty. To this end, they obtain samples of the uncertain parameter values, and evaluate the robustness of a software system under uncertainty by running Monte-Carlo simulations that use these empirical probability distributions.

More recently, Dorn et al. (2020) focuses on understanding the influence of configuration options on performance and proposes an approach based on probabilistic programming that explicitly models uncertainty for option influences and provides both a scalar and a confidence interval for each prediction of a configuration's performance. A method that uses mathematical formulas for incorporating and evaluating epistemic uncertainty of the input parameters of queueing models is presented in Antonelli et al. (2020). A similar approach to the study of uncertainty propagation in reliability models is introduced in Mishra and Trivedi (2013).

A different philosophy in dealing with uncertainty involves the adoption of self-adaptation in software systems. The number of studies that consider uncertainty in self-adaptive systems has increased in recent years. A typical example is the use of probabilistic run-time models, such as Markov decision processes (Calinescu et al., 2017a; Moreno et al., 2015) and parametric stochastic models (Calinescu et al., 2018) to reason about uncertainty and change when making adaptation decisions. The approaches introduced in Weyns et al. (2015) and Musil et al. (2017) employ self-adaptation to cope with uncertainty. The approach from Weyns et al. (2015) proposes a combination of adaptation and evolution of the software to make its behaviour resilient to uncertainty, which in turn entails that the software system is *sustainable*, while Musil et al. (2017) focuses on the uncertainty surrounding the execution of cyber–physical production systems. A different approach can be found in Shevtsov et al. (2019), where a control-theoretic approach is adopted to handle uncertainty in self-adaptive software systems. Furthermore, the need for software systems to operate well under the existing uncertainties is among the main waves that have advanced the research on self-adaptive systems (Weyns, 2017), although a *perpetual assurance* of goal satisfaction in self-adaptive systems is still an open research challenge (Weyns et al., 2017). Most of these results consider uncertainty in the decision-making process and propose adaptation approaches that guarantee the quality requirements under different and (possibly) unknown types of changes.

Our work lies in the area of the reduction of parametric epistemic uncertainty and introduces an adaptive uncertainty reduction heuristic for performance and dependability software engineering. The proposed heuristic is integrated into a new iterative approach that exploits the adoption of formal verification with confidence intervals.

One of the key aspects of the proposed approach consists of the identification of the system component for which additional data – to be obtained through testing – are needed. The selection of components to be tested in each iteration is based on a combination of factors that include the sensitivity of the model to variations in the parameters of different components, and the overheads of unit-testing each of these components. Reducing the cost of the (reliability) testing phase by selecting key components to test is a topic that has been analysed in the literature. For example, Garg et al. (2011) tackles the question "When to stop testing" by focusing on reliability and discussing the challenges and the potentials related to existing software reliability models. Classical approaches in this domain are based on operational profile (Musa, 1996), however operational profile is often unknown and subject to changes. To overcome this problem, Cotroneo et al. (2016) proposes an adapting testing schema that iteratively learns from test execution results as they become available, and, based on them, allocates test cases to the most sensible parts. The assessment is then performed adopting a second sampling strategy that provides the interval estimate of the reliability computed during testing. A different approach that focuses on the allocation of testing resources under uncertain conditions is presented in Pietrantuono et al. (2018). In this approach, a multi-objective debug-aware and robust optimisation problem under uncertainty of data is used to evaluate of alternative trade-offs among reliability, cost, and release time.

Compared to these approaches, VERACITY has the major advantage that it uses formal quantitative verification to compute confidence intervals for the relevant nonfunctional properties of the system under verification. As such, our approach mitigates the risk of generating inaccurate single-point estimates for these properties. Furthermore, its use of Markov models makes VERACITY applicable to multiple classes of systems for which such models are extensively used, as detailed in Section 6.5.

8. Conclusion

We presented VERACITY, a tool-supported approach for the efficient verification of nonfunctional requirements under uncertainty. VERACITY operates by acquiring information about the components of the verified system through testing them individually over a number of verification rounds. A user-defined testing budget specifies the amount of testing performed in each round, and the partition of this budget among system components is adapted from one round to the next in order to complete the verification process with a low overall testing cost. The heuristic used to compute this adaptive partition considers factors such as the sensitivity of the verified requirements to the parameters associated with different components, and the different cost (e.g., time, price or risk) of testing these components. The evaluation of VERACITY in case studies from the areas of service-based systems and web applications showed that, on average, it significantly reduces the overall testing cost required to complete the verification process compared to uniformly partitioning the testing budget across all system components.

In future work, we plan to expand the set of factors underpinning our VERACITY round-budget partitioning, in order to further improve its efficiency. One such additional factor that we are considering is the level of epistemic uncertainty associated with each component: in each round, a larger fraction of the testing budget should be allocated to components with higher levels of epistemic uncertainty, i.e., to those for which fewer observations are already available. We envisage that augmenting our heuristic with this factor will extend the applicability of VERACITY to verification scenarios in which observations about a subset of the system components are already available at the beginning of the verification process (e.g., from previous testing of those components), and we plan to carry out additional case studies to validate this hypothesis.

Another important direction of future research for our project is to consider the scenarios in which (i) the imbalance between the component testing costs is much higher than the 1:5 ratio considered in our experiments so far; and/or (ii) some of the parameters that the verified requirements depend on are associated with the operational profile of the system, i.e., with parameters whose epistemic uncertainty cannot be lowered by testing the components of the system. For the second scenario, examples of such parameters include the number of requests received by a web server in one hour, and the probabilities of these requests being of different types. To some extent, VERACITY could handle this scenario by associating such parameters with an "operational profile component" that is assigned an infinite testing cost. Because this "component" will never be tested, the verification problem may be undecidable, in which case VERACITY will (correctly) terminate with a 'budget exhausted' outcome. However, this outcome will only be produced after significant testing effort, some of which could be avoidable by noticing – before using all the testing budget – that the operational profile uncertainty renders the verification problem undecidable. We plan to extend VERACITY with the ability to report an 'undecidable' outcome (without exhausting the testing budget) in this important verification scenario.

Last but not least, the verification problem tackled by VERACITY can be generalised in multiple ways. For example, cost can be considered a multi-dimensional entity with separate elements for time, monetary cost, etc.; in such a case, the budget would also be a tuple with these elements. As another example, it may be of interest to use different confidence levels $\alpha_1, \alpha_2, \ldots, \alpha_n$ for the n requirements from (4). To handle this variant of the problem, we envisage that VERACITY will need to be augmented with the ability to acquire more observations for the system components with parameters that influence the requirements associated with high confidence levels than for the other components. We plan to explore this hypothesis in our future work.

CRediT authorship contribution statement

Naif Alasmari: Conceptualisation, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualisation. **Radu Calinescu:** Conceptualisation, Methodology, Writing – original draft, Writing – review & editing, Visualisation, Supervision, Funding acquisition. **Colin Paterson:** Formal analysis, Writing – original draft, Visualisation. **Raffaela Mirandola:** Conceptualisation, Methodology, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This project has received funding from the Assuring Autonomy International Programme, UK, the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience', and the ORCA-Hub PRF project 'COVE'.

Appendix

Markov chain construction from UML activity diagrams. A parametric discrete-time Markov chain (1) can be derived from the UML activity diagram of a software system by following the step-by-step process summarised below:

1. Construct the state set S consisting of a state for each activity node from the UML activity diagram, plus an initial state s_0 and an "end" state s_{end} associated with the initial and final nodes of the activity diagram, respectively. For each state $s \in S$, let $node(s)$ represent the activity diagram node corresponding to state s.
2. Set $\mathbf{P}(s, s') = 1.0$ for every pair of states $s, s' \in S$ for which the node reached immediately after $node(s)$ in the activity diagram (i.e., without traversing activity diagram nodes associated with states from $S \setminus \{s, s'\}$) is *always* $node(s')$.
3. Set $\mathbf{P}(s_{\mathrm{end}}, s_{\mathrm{end}}) = 1.0$.
4. Associate an unknown transition probability $\mathbf{P}(s, s')$ with each pair of states $s, s' \in S$ for which $node(s')$ can be reached from $node(s)$ by traversing only decision nodes from the activity diagram.
5. Set $\mathbf{P}(s, s') = 0$ for every other pair of states $s, s' \in S$.
6. Assemble the atomic proposition set

   $$AP = \{start, end\} \cup \{name(node(s)) \mid s \in S \setminus \{s_0, s_{\mathrm{end}}\}\}$$

   where $name(node(s))$ is a unique name for the activity node $node(s)$.
7. Set $L(node(s)) = name(node(s))$ for every state $s \in S \setminus \{s_0, s_{\mathrm{end}}\}$, $L(s_0) = start$, and $L(s_{\mathrm{end}}) = end$.

Further details about this process are available in Calinescu and Rafiq (2013) and Ghezzi et al. (2013).

References

Afzal, H., Mufti, M.R., Awan, I., Yousaf, M., 2019. Performance analysis of radio spectrum for cognitive radio wireless networks using discrete time Markov chain. J. Syst. Softw. 151, 1–7.

Akobeng, A.K., 2008. Confidence intervals and p-values in clinical decision making. Acta Pædiatr. 97 (8), 1004–1007.

Ameller, D., Franch, X., Gómez, C., Martínez-Fernández, S., Araujo, J., Biffl, S., Cabot, J., Cortellessa, V., Méndez, D., Moreira, A., Muccini, H., Vallecillo, A., Wimmer, M., Amaral, V., Bühm, W., Bruneliere, H., Burgueño, L., Goulão, M., Teufl, S., Berardinelli, L., 2019. Dealing with non-functional requirements in model-driven development: A survey. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2019.2904476.

Andova, S., Hermanns, H., Katoen, J.-P., 2004. Discrete-time rewards model-checked. In: International Conference on Formal Modeling and Analysis of Timed Systems. Springer, pp. 88–104. http://dx.doi.org/10.1007/978-3-540-40903-8_8.

Antonelli, F., Cortellessa, V., Gribaudo, M., Pinciroli, R., Trivedi, K.S., Trubiani, C., 2020. Analytical modeling of performance indices under epistemic uncertainty applied to cloud computing systems. Future Gener. Comput. Syst. 102, 746–761. http://dx.doi.org/10.1016/j.future.2019.09.006.

Aughenbaugh, J.M., Paredis, C.J., 2006. The value of using imprecise probabilities in engineering design. J. Mech. Des. 128 (4), 969–979.

Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P., 2007. Validation of web service compositions. IET Softw. 1 (6), 219–232. http://dx.doi.org/10.1049/iet-sen:20070027.

Barlow, R.E., Proschan, F., 1975. Importance of system components and fault tree events. Stochastic Process. Appl. 3 (2), 153–173.

Benjamin, J.R., Cornell, C.A., 2014. Probability, Statistics, and Decision for Civil Engineers. Courier Corporation.

Birnbaum, Z.W., 1969. On the importance of different components in a multicomponent system. In: Krishnaiah, P.R. (Ed.), Multivariate Analysis II. Academic Press, New York.

Calinescu, R., Autili, M., Cámara, J., Di Marco, A., Gerasimou, S., Inverardi, P., Perucci, A., Jansen, N., Katoen, J.-P., Kwiatkowska, M., Mengshoel, O.J., Spalazzese, R., Tivoli, M., 2017a. Synthesis and verification of self-aware computing systems. In: Kounev, S., Kephart, J.O., Milenkoski, A., Zhu, X. (Eds.), Self-Aware Computing Systems. Springer, pp. 337–373.

Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N., 2018. Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. 143, 140–158.

Calinescu, R., Ghezzi, C., Johnson, K., Pezzé, M., Rafiq, Y., Tamburrelli, G., 2015. Formal verification with confidence intervals to establish quality of service properties of software systems. IEEE Trans. Reliab. 65 (1), 107–125.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G., 2011. Dynamic QoS management and optimization in service-based systems. IEEE Trans. Softw. Eng. 37 (3), 387–409.

Calinescu, R., Johnson, K., Paterson, C., 2016. FACT: A probabilistic model checker for formal verification with confidence intervals. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp. 540–546.

Calinescu, R., Johnson, K., Rafiq, Y., 2013. Developing self-verifying service-based systems. In: 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE, pp. 734–737. http://dx.doi.org/10.1109/ASE.2013.6693145.

Calinescu, R., Paterson, C.A., Johnson, K., 2021. Efficient parametric model checking using domain knowledge. IEEE Trans. Softw. Eng. 47 (6), 1114–1133. http://dx.doi.org/10.1109/TSE.2019.2912958.

Calinescu, R., Rafiq, Y., 2013. Using intelligent proxies to develop self-adaptive service-based systems. In: 2013 International Symposium on Theoretical Aspects of Software Engineering. IEEE, pp. 131–134.

Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T., 2017b. Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Trans. Softw. Eng. 44 (11), 1039–1069.

Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., 2012. Non-Functional Requirements in Software Engineering, Vol. 5. Springer Science & Business Media.

Ciesinski, F., Größer, M., 2004. On probabilistic computation tree logic. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (Eds.), Validation of Stochastic Systems: A Guide to Current Research. Springer, pp. 147–188. http://dx.doi.org/10.1007/978-3-540-24611-4_5.

Cotroneo, D., Pietrantuono, R., Russo, S., 2016. RELAI testing: A technique to assess and improve software reliability. IEEE Trans. Softw. Eng. 42 (5), 452–475. http://dx.doi.org/10.1109/TSE.2015.2491931.

Daka, E., Fraser, G., 2014. A survey on unit testing practices and problems. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering. pp. 201–211. http://dx.doi.org/10.1109/ISSRE.2014.11.

Daws, C., 2005. Symbolic and parametric model checking of discrete-time Markov chains. In: International Colloquium on Theoretical Aspects of Computing. pp. 280–294. http://dx.doi.org/10.1007/978-3-540-31862-0_21.

Dehnert, C., Junges, S., Katoen, J.-P., Volk, M., 2017. A storm is coming: A modern probabilistic model checker. In: 29th International Conference on Computer Aided Verification (CAV). pp. 592–600. http://dx.doi.org/10.1007/978-3-319-63390-9_31.

Dorn, J., Apel, S., Siegmund, N., 2020. Mastering uncertainty in performance estimations of configurable software systems. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 684–696.

du Prel, J.-B., Hommel, G., Röhrig, B., Blettner, M., 2009. Confidence interval or p-value? (part 4 of a series on evaluation of scientific publications). Deutsch. Ärztebl. Int. 106 (19), 335.

Esfahani, N., Malek, S., 2013. Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II. In: Lecture Notes in Computer Science, vol. 7475, Springer, pp. 214–238. http://dx.doi.org/10.1007/978-3-642-35813-5_9.

Fabijan, A., Dmitriev, P., McFarland, C., Vermeer, L., Holmström Olsson, H., Bosch, J., 2018. Experimentation growth: Evolving trustworthy A/B testing capabilities in online software companies. J. Softw.: Evol. Process 30 (12), e2113.

Fang, X., Calinescu, R., Gerasimou, S., Alhwikem, F., 2021. Fast parametric model checking through model fragmentation. In: 43rd IEEE/ACM International Conference on Software Engineering. pp. 835–846.

Filieri, A., Ghezzi, C., Tamburrelli, G., 2012. A formal approach to adaptive software: continuous assurance of non-functional requirements. Form. Asp. Comput. 24 (2), 163–186.

Filieri, A., Pasareanu, C.S., Visser, W., 2013. Reliability analysis in symbolic pathfinder. In: 35th International Conference on Software Engineering (ICSE). IEEE, pp. 622–631. http://dx.doi.org/10.1109/ICSE.2013.6606608.

Franco, J.M., Correia, F., Barbosa, R., Zenha-Rela, M., Schmerl, B., Garlan, D., 2016. Improving self-adaptation planning through software architecture-based stochastic modeling. J. Syst. Softw. 115, 42–60.

Gallotti, S., Ghezzi, C., Mirandola, R., Tamburrelli, G., 2008. Quality prediction of service compositions through probabilistic model checking. In: International Conference on the Quality of Software Architectures. Springer, pp. 119–134.

Gardner, M.J., Altman, D.G., 1986. Confidence intervals rather than p values: estimation rather than hypothesis testing. Br. Med. J. 292 (6522), 746–750. http://dx.doi.org/10.1136/bmj.292.6522.746.

Garg, M., Lai, R., Huang, S.J., 2011. When to stop testing: a study from the perspective of software reliability models. IET Softw. 5, 263–273, (10). URL https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2010.0007.

Garlan, D., 2010. Software engineering in an uncertain world. In: Future of Software Engineering Research. ACM, pp. 125–128.

Gerasimou, S., Calinescu, R., Tamburrelli, G., 2018. Synthesis of probabilistic models for quality-of-service software engineering. Autom. Softw. Eng. 25 (4), 785–831.

Ghezzi, C., Pinto, L.S., Spoletini, P., Tamburrelli, G., 2013. Managing non-functional uncertainty via model-driven adaptivity. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE, pp. 33–42.

Ghezzi, C., Sharifloo, A.M., 2013. Model-based verification of quantitative non-functional properties for software product lines. Inf. Softw. Technol. 55 (3), 508–524.

Giese, H., Bencomo, N., Pasquale, L., Ramirez, A.J., Inverardi, P., Wätzoldt, S., Clarke, S., 2014. Living with uncertainty in the age of runtime models. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (Eds.), Models@Run.Time: Foundations, Applications, and Roadmaps. Springer International Publishing, Cham, pp. 47–100.

Grissom, R.J., Kim, J.J., 2005. Effect Sizes for Research: A Broad Practical Approach. Lawrence Erlbaum Associates Publishers.

Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L., 2010. PARAM: A model checker for parametric markov models. In: Computer Aided Verification. Springer, pp. 660–664. http://dx.doi.org/10.1007/978-3-642-14295-6_56.

Hansson, H., Jonsson, B., 1994. A logic for reasoning about time and reliability. Form. Asp. Comput. 6 (5), 512–535. http://dx.doi.org/10.1007/BF01211866.

Johnson, K., Calinescu, R., Kikuchi, S., 2013. An incremental verification framework for component-based software systems. In: The 16th International ACM Sigsoft Symposium on Component-Based Software Engineering. ACM, pp. 33–42.

Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I., 2007. A systematic review of effect size in software engineering experiments. Inf. Softw. Technol. 49 (11–12), 1073–1086.

Kirkwood, B.R., Sterne, J.A., 2010. Essential Medical Statistics. John Wiley & Sons.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A., 2017. Robust statistical methods for empirical software engineering. Empir. Softw. Eng. 22 (2), 579–630.

Kohavi, R., Longbotham, R., 2017. Online controlled experiments and A/B testing. Encycl. Mach. Learn. Data Min. 7 (8), 922–929.

Krka, I., Golubchik, L., Medvidovic, N., 2010. Probabilistic automata for architecture-based reliability assessment. In: ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems. pp. 17–24. http://dx.doi.org/10.1145/1808877.1808881.

Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification. Springer, pp. 585–591. http://dx.doi.org/10.1007/978-3-642-22110-1_47.

Llerena, Y.R.S., Böhme, M., Brünink, M., Su, G., Rosenblum, D.S., 2018. Verifying the long-run behavior of probabilistic system models in the presence of uncertainty. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 587–597.

Madeyski, L., 2009. Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer Science & Business Media.

Madeyski, L., Kitchenham, B., 2018. Effect sizes and their variance for ab/ba crossover design studies. Empir. Softw. Eng. 23 (4), 1982–2017.

Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G., 1998. Modelling with generalized stochastic Petri nets. ACM SIGMETRICS Perform. Eval. Rev. 26 (2), 2. http://dx.doi.org/10.1145/288197.581193.

Meedeniya, I., Aleti, A., Grunske, L., 2012. Architecture-driven reliability optimization with uncertain model parameters. J. Syst. Softw. 85 (10), 2340–2355.

Meedeniya, I., Moser, I., Aleti, A., Grunske, L., 2011. Architecture-based reliability evaluation under uncertainty. In: The Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS. ACM, pp. 85–94. http://dx.doi.org/10.1145/2000259.2000275.

Mishra, K., Trivedi, K.S., 2013. Closed-form approach for epistemic uncertainty propagation in analytic models. In: Dohi, T., Nakagawa, T. (Eds.), Stochastic Reliability and Maintenance Modeling: Essays in Honor of Professor Shunji Osaki on his 70th Birthday. Springer London, London, pp. 315–332. http://dx.doi.org/10.1007/978-1-4471-4971-2_14.

Moreno, G.A., Camara, J., Garlan, D., Schmerl, B., 2015. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In: Foundations of Software Engineering. ACM, pp. 1–12.

Muccini, H., Dias, M., Richardson, D.J., 2006. Software architecture-based regression testing. J. Syst. Softw. 79 (10), 1379–1396. http://dx.doi.org/10.1016/j.jss.2006.02.059, architecting Dependable Systems. URL https://www.sciencedirect.com/science/article/pii/S0164121206001361.

Musa, J.D., 1996. The operational profile. In: Özekici, S. (Ed.), Reliability and Maintenance of Complex Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 333–344.

Musil, A., Musil, J., Weyns, D., Bures, T., Muccini, H., Sharaf, M., 2017. Patterns for self-adaptation in cyber–physical systems. In: Multi-Disciplinary Engineering for Cyber–Physical Production Systems. Springer, pp. 331–368.

Nakagawa, H., Toyama, H., Tsuchiya, T., 2019. Expression caching for runtime verification based on parameterized probabilistic models. J. Syst. Softw. 156, 300–311.

Paterson, C., Calinescu, R., 2020. Observation-enhanced QoS analysis of component-based systems. IEEE Trans. Softw. Eng. 46 (05), 526–548. http://dx.doi.org/10.1109/TSE.2018.2864159.

Perez-Palacin, D., Mirandola, R., 2014. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: 5th ACM/SPEC International Conference on Performance Engineering. ACM, pp. 3–14. http://dx.doi.org/10.1145/2568088.2568095.

Perez-Palacin, D., Mirandola, R., Merseguer, J., 2012. QoS and energy management with Petri nets: A Self-adaptive framework. J. Syst. Softw. 85 (12), 2796–2811. http://dx.doi.org/10.1016/j.jss.2012.04.077.

Pietrantuono, R., Potena, P., Pecchia, A., Rodríguez, D., Russo, S., Sanz, L.F., 2018. Multiobjective testing resource allocation under uncertainty. IEEE Trans. Evol. Comput. 22 (3), 347–362. http://dx.doi.org/10.1109/TEVC.2017.2691060.

Ramirez, A.J., Jensen, A.C., Cheng, B.H.C., 2012. A taxonomy of uncertainty for dynamically adaptive systems. In: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. In: SEAMS '12, IEEE Press, Piscataway, NJ, USA, pp. 99–108, URL http://dl.acm.org/citation.cfm?id=2666795.2666812.

Shevtsov, S., Weyns, D., Maggio, M., 2019. SimCA*: A control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees. ACM Trans. Auton. Adapt. Syst. 13 (4), 1–34. http://dx.doi.org/10.1145/3328730.

Siroker, D., Koomen, P., 2013. A/B Testing: The Most Powerful Way to Turn Clicks into Customers. John Wiley & Sons.

Su, G., Feng, Y., Chen, T., Rosenblum, D.S., 2016a. Asymptotic perturbation bounds for probabilistic model checking with empirically determined probability parameters. IEEE Trans. Softw. Eng. 42 (7), 623–639. http://dx.doi.org/10.1109/TSE.2015.2508444.

Su, G., Rosenblum, D.S., Tamburrelli, G., 2016b. Reliability of run-time quality-of-service evaluation using parametric model checking. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 73–84.

The European Mathematical Society, 2020. The Encyclopedia of Mathematics. URL https://encyclopediaofmath.org/wiki/Rational_function.

Trubiani, C., Meedeniya, I., Cortellessa, V., Aleti, A., Grunske, L., 2013. Model-based performance analysis of software architectures under uncertainty. In: The 9th International ACM Sigsoft Conference on Quality of Software Architectures. ACM, pp. 69–78. http://dx.doi.org/10.1145/2465478.2465487.

Walker, W., Harremoes, P., Romans, J., van der Sluus, J., van Asselt, M., Janssen, P., Krauss, M., 2003. Defining uncertainty. a conceptual basis for uncertainty management in model-based decision support. Integr. Assess. 4 (1), 5–17.

Wang, X., Sun, J., Chen, Z., Zhang, P., Wang, J., Lin, Y., 2018. Towards optimal concolic testing. In: Proceedings of the 40th International Conference on Software Engineering. pp. 291–302.

Weyns, D., 2017. Software engineering of self-adaptive systems: an organised tour and future challenges. In: Chapter in Handbook of Software Engineering. Springer.

Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.-M., Malek, S., et al., 2017. Perpetual assurances for self-adaptive systems. In: Software Engineering for Self-Adaptive Systems III. Assurances. Springer, pp. 31–63.

Weyns, D., Caporuscio, M., Vogel, B., Kurti, A., 2015. Design for sustainability = runtime adaptation ∪ evolution. In: The 2015 European Conference on Software Architecture Workshops. ACM, pp. 62:1–62:7. http://dx.doi.org/10.1145/2797433.2797497.

Naif Alasmari is a Ph.D. student in the Department of Computer Science at the University of York, UK. His research interests are in quantitative verification of component-based systems, verification under uncertainty, and correct-by-construction synthesis of discrete-event controllers.

Radu Calinescu is Professor of Computer Science at the University of York, UK. His main research interests are in formal methods for self-adaptive, autonomous, secure and resilient autonomous and AI systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive and autonomous systems and processes.

Colin Paterson is a Senior Postdoctoral Researcher in Computer Science at the University of York, UK. He holds Ph.D.s in Control Systems Engineering, gained in collaboration with Jaguar Cars, and Computer Science, in which techniques for the verification of operational processes, using observation data, were developed. His current research considers techniques for the development and assurance of resilient autonomous systems operating in uncertain environments.

Raffaela Mirandola is Associate Professor in the Dipartimento di Elettronica, Informazione e Bioingegneria at Politecnico di Milano, Italy. Her research interests include the areas of performance and reliability modelling and analysis of software/hardware systems with special emphasis on: methods for the automatic generation of performance and reliability models for component-based and service-based systems, and methods to develop software that is dependable and can easily evolve, possibly self-adapting its behaviour.