The Journal of Systems and Software 161 (2020) 110462

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

On the testing resource allocation problem: Research trends and perspectives

Roberto Pietrantuono

Università degli Studi di Napoli Federico II, Via Claudio 21, Naples 80125, Italy

ARTICLE INFO

Article history:
Received 19 March 2019
Revised 8 November 2019
Accepted 12 November 2019
Available online 14 November 2019

Keywords:
Testing
Resource allocation
Reliability allocation
Literature review
Test planning
Survey

ABSTRACT

In testing a software application, a primary concern is how to effectively plan the assignment of resources available for testing to the software components so as to achieve a target goal under given constraints. In the literature, this is known as *testing resources allocation problem* (TRAP). Researchers spent a lot of effort to propose models for supporting test engineers in this task, and a variety of solutions exist to assess the best trade-off between testing time, cost and quality of delivered products. This article presents a systematic mapping study aimed at systematically exploring the TRAP research area in order to provide an overview on the type of research performed and on results currently available. A sample of 68 selected studies has been classified and analyzed according to defined dimensions. Results give an overview of the state of the art, provide guidance to improve practicability and allow outlining a set of directions for future research and applications of TRAP solutions.

1. Introduction

Testing is a crucial activity of software development, which greatly impacts the quality products as well as the production cost and time-to-market. More often than not, engineers committed on testing are required to complete the testing process on time (often working with underestimated resources), while assuring a high quality of the product to ship.

Although the experience of engineers is paramount for an efficient test planning, the problem of how to optimally use the available resources for testing cannot rely solely on that, but demands for a systematic approach. This is especially true in large/complex systems and development processes, with many contrasting objectives and constraints on quality, time and cost. A wrong planning can end up in serious time/budget overrun or in a bad quality of delivered products.

A rough-grained approach is to allocate a greater effort to software modules expected to contain more faults,[1] namely, by exploiting the so-called *fault-proneness models*. These models are conceived to assess the defectiveness of software modules by exploiting process or product metrics as features of a classifier/regressor (such as *code-level* metrics, e.g., cyclomatic complexity, lines of code, object-oriented metrics, fan-in, fan-out; *file-level* metrics from the CVS/SVN/Git repositories; *design* metrics; *requirements* metrics): They apply common machine learning algorithms on instances with a known number of faults to train the classifier/regressor, and use the trained model to predict fault-proneness in modules with unknown number of faults. The output is either the list of modules believed to be fault-prone (namely, containing at least one fault) and fault-free — in a binary formulation — or the rank of modules from the most to the least fault-prone one — when formulated as a ranking problem (Yang et al., 2015b). There is a huge literature on fault-proneness modules — see these surveys for reference (Malhotra, 2015; Hall et al., 2012; Catal and Diri, 2009).

The problem with this formulation is that the output does not directly tell how to allocate testing resources to modules, i.e., there is no allocation scheme following the classification/ranking task. Clearly, fault-prone modules deserve more testing resources, but the amount of resources is not quantified (apart from few exceptions, e.g. (Monden et al., 2013)). For a finer-granied and more general approach, researchers formulated the testing resource allocation as an optimization problem. In a *testing resource allocation problem* (TRAP), the solution suggests the exact amount of effort to

E-mail address: roberto.pietrantuono@unina.it

[1] Note that the software engineering literature typically uses the term *defect*, while in reliability research the term *fault* is used, according to the *fault-error-failure chain* definition (Avizienis et al., 2004): in this work we use them synonymously to denote the adjudged cause of an observed failure.

https://doi.org/10.1016/j.jss.2019.110462
0164-1212/

spent for testing each module in order to attain one or more objectives (e.g., in terms of number of detected defects, user-perceived reliability,[2] testing cost, testing time) under given constraints.

Several alternative formulations for the TRAP have been explored by researchers. Over the years, the increasing awareness of the many factors that potentially affect the effectiveness of an allocation plan, as well as the availability of powerful solution methods such as metaheuristics, have paved the way to more and more accurate (and complex) models. Today, researchers actively work on proposing formulations able to accomodate the needs emerging with new testing scenarios.

This article provides an overview on the studies dealing with the testing resource allocation problem. Few researchers have reviewed, by secondary studies, the state-of-the-art of some testing problems related, though loosely, to the TRAP. Elberzhager et al. reviewed approaches for testing effort estimation and reduction (Elberzhager et al., 2012); Calp et al. recently surveyed the area of test planning activities (CALP and KÖSE, 2018); Lee et al. looked at existing testing practices in industry (Lee et al., 2012). However, none of them target the TRAP literature. As confirmed by a recent tertiary study on software testing (Garousi and Mäntylä, 2016), the area of test management, which is the closest area to TRAP, still lacks secondary studies.

In this article, the systematic mapping (SM) study method is applied in order to identify, classify and evaluate the current state of the art from the following perspectives: *publication trends*, to figure out how researchers are committed on this topic; *formulation*, to explore the several alternatives for setting up a TRAP model; *solution*, to understand how a TRAP model is typically solved; *validation*, to highlight how models and solutions are validated in TRAP research. The study focuses on a set of 68 selected papers. A classification is defined in order to categorize the studies and ease the analysis. The results obtained from synthesized data produce a clear overview of the state of the art, allow outlining guidelines for supporting the practice of TRAP strategies, and give a solid basis to plan for future research of TRAP solutions.

The article is structured as follows. Section 2 provides background on the TRAP. Section 3 describes the design of the study in line with the *systematic mapping* criteria. Section 4 reports the results according to categories defined in Section 3. Section 5 elaborates on the gaps of existing research and on potential directions for future research. Section 6 provides guidelines on how the availalable results on existing TRAP approaches can be applied in practice. Section 7 warns against threats to validity and Section 8 concludes the article.

2. The testing resource allocation problem

2.1. Overview

The problem of identifying which parts of a system should receive more resources[3] has been addressed by many researchers. The TRAP can be seen as an instance of the more general *Reliability-Redundancy Allocation Problem* (R-RAP) (Kuo and Wan, 2007). An R-RAP combines two common ways to improve reliability of a system: *i)* Providing redundant components in parallel and *ii)* improving the individual system's component reliability[4] (Kuo et al., 2001). Its solution suggests the choice of components in an architecture with their redundancy level and/or how much effort per component has to be spent, in order to either maximize the system reliability under a budget constraint or, conversely, minimize the cost that satisfies a minimum demand on system reliability. An R-RAP can refer to any phase of the development cycle, from design to testing — hence "resources" to allocate generically refers to any means to improve reliability (e.g., increase redundancy, buy a component rather than another, spend more design effort or testing effort). These problems have been well-developed for many different system structures (e.g., in series-parallel structures), objective functions, redundancy strategies, and time-to-failure distributions. A lot of work considered the R-RAP especially related to the *design phase* of a system (e.g., for the evaluation of architectural alternatives), both referring to hardware systems (e.g., Nakagawa and Miyazaki, 1981; Tillman et al., 1977) and to software systems (e.g., Elegbede et al., 2003; Rice et al., 1999).

The TRAP is a particular R-RAP that refers to the *testing phase* rather than to the design phase — thus, resources to distribute refer to the *testing effort* needed to improve components' reliability. In TRAPs, allocating testing effort to a component is the main way to improve its reliability. Other means, such as fault tolerance or redundancy, can still be considered in the formulation, usually as variables in the constraints rather than as decision variables in the objective functions. In its typical formulation, the TRAP is referred to the *architectural* level, namely it is about how to allocate resources for testing to components of a system. It should be noted that researchers often do not specify what they meant by "component" (in some works they refer generically to "modules" or modular systems; in other cases components are defined as logically independent units performing a well-defined function, like in component-based models such as CCM, EJB or DCOM; in others they refer to physical components; or it is simply not specified), but the underlying assumption is components are units that can be tested independently (hence, testing resources can be partitioned between them). The common objectives in a typical TRAP are about the system *reliability*, the testing *cost*, the testing *effort* or *time*. These are either contrasted to each other (in a multi-objective optimization) with the aim of finding suitable trade-offs, or one of them is set as objective constrained on the others (in a single-objective optimization). The general form of a single-objective TRAP model looks like one of the following:

$$
\begin{aligned}
&max \quad R_S(x|T_1, T_2, \ldots, T_n) \qquad s.t. \qquad T = \sum_{i=1}^{n} T_i \leq T^* \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad and/or \quad C = \sum_{i=1}^{n} C_i \leq C^* \\
&min \quad T = \sum_{i=1}^{n} T_i \qquad\qquad s.t. \qquad R_S(x|T_1, T_2, \ldots .T_n) \geq R^* \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad and/or \quad C = \sum_{i=1}^{n} C_i \leq C^* \\
&min \quad C = \sum_{i=1}^{n} C_i \qquad\qquad s.t. \qquad R_S(x|T_1, T_2, \ldots .T_n) \geq R^* \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad and/or \quad \sum_{i=1}^{n} T_i \leq T^*
\end{aligned}
\tag{1}
$$

where:

- R_S is the overall reliability of the software (or related functions, e.g., expected failure intensity, that is number of failure per time unit);
- T is the total testing effort, and
- T_i is the effort for testing component i (out of n components);

[2] Reliability is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment (Lyu, 1996)

[3] Here we refer to a *component* or *module* as an independently testable functionality. The terms are used as synonymous if not differently specified.

[4] When only redundancy is employed, the problem is referred to as Redundancy Allocation Problem (RAP))

C is the total testing cost;
C_i is the cost for testing component i;
T^* is the maximum testing effort;
C^* is the maximum testing cost;
R^* is the minimum level of required reliability.

The multi-objective version considers two or more of these objectives together. It should be noted that these functions are interdependent. For instance, the achieved reliability of a module depends on the testing effort devoted to it; the cost depends on testing effort (e.g., including the cost of tester) but can also account for the cost of debugging during testing or during operation. Thus, formulations vary based on the models adopted to capture such inter-dependencies. When both the testing cost and time/effort are taken into account, a possible issue to consider is multicollinearity, caused by the tight relation between these two objectives. The extent of such a problem, hence the need for remedies, is dependent on how the cost and time/effort functions are modelled. The issue can affect the accuracy of the output predictions, but it is still mostly neglected in multi-objective TRAP studies. Finally, there are different ways of expressing inter-component architectural dependencies, which determine how an attribute of interest at system level (e.g., reliability) is computed from the same attribute at component level. In the next Section, a brief overview of common objective functions is given.

2.2. Common objective functions

Most TRAP studies exploit *Software Reliability Growth Models* (SRGMs) as a way to capture the relation between the *reliability growth* and the *testing effort (or time)* devoted to each module, which are two objectives occurring almost always together. Testing a system implies exercising the program with a set of test cases, observing the output, and comparing it with the expected one such that if they are discordant, a *failure* is said to have occurred. The adjudged cause of a failure is a fault: in SRGMs, testing is modeled as a *fault detection and correction* process, during which more faults are detected (namely, failures are observed) as more testing effort is spent and reliability progressively grows as such detected faults are corrected. SRGMs are a wide class of models fitting inter-failure times observed during testing and debugging to predict the next time to failure. They capture the reliability growth as software is improved by faults detection and correction and differ from each other in the shape of the fault detection/correction curve. There is a huge literature on SRGMs: Since the end of the 70s, researchers constantly look for analytical forms to faithfully model the fault detection/correction process (FDP/FCP) of new types of applications and testing processes. The most common class of SRGMs is the Non-Homogeneous Poisson Process (NHPP) SRGMs. In NHPP SRGMs, the mean number of faults detected in the time interval $(t, t+\Delta t]$ is assumed proportional to the mean number of residual faults. This proportionality is expressed by the fault detection rates per fault as functions of time, denoted with $\lambda(t)$. From this, the mean value function of the fault detection process is expressed as:

$$\frac{dm_d(t)}{dt} = \lambda(t)(\alpha - m_d(t)), \quad \alpha > 0 \tag{2}$$

where α is the estimated initial number of faults. In case of constant fault detection rate ($\lambda(t) = \beta$), $m_d(t)$ is the Goel–Okumoto exponential SRGM ($m_d(t) = \alpha(1 - e^{-\beta t})$).

It can be shown that using Eq. (2) to describe the fault detection processes, and defining $D(t) = \int_0^t \lambda(s)ds$, the cumulative number of detected faults is given by Lo and Huang (2006):

$$m_d(t) = a\left(1 - e^{-D(t)}\right) \tag{3}$$

where:

$m_d(t)$ is the cumulative number of detected and corrected faults at time t;
$D(t)$ models the Fault Detection Process (FDP) by means of the *fault detection rate per remaining fault* λ (also called failure intensity): $D(t) = \int_0^t \lambda(s)ds$;
a is the expected number of total faults.

The shape of $D(t)$ determines the specific SRGM (e.g., exponential, S-Shaped, Logarithmic, Log-Logistic).

Examples are the exponential Goel and Okumoto (GO) model and its generalised version (Goel and Okumoto, 1979),(Goel, 1985); the S-Shaped model by Yamada, conceived to capture the possible increase/decrease of the fault detection rate during testing (Yamada et al., 1983); the Gokhale and Trivedi log-logistic model, that also follows an increasing/decreasing pattern describing the initial phase of testing as characterised by a slow learning phase (Gokhale and Trivedi, 1998); the Gompertz SRGM, by Ohishi et al., derived from the statistical theory of extreme-value (Ohishi et al., 2009).

Eq. 3 captures the FDP but neglects the *fault correction process* (FCP), namely it assumes an *immediate* debugging time, which however can have a severe impact on the estimation accuracy (Cinque et al., 2017), (Cinque et al., 2014). Similarly to what described above, the mean number of faults corrected in $(t, t+\Delta t]$ is assumed to be proportional to the mean number of detected but not yet corrected faults. This proportionality is expressed by the fault correction rates per fault as function of time, denoted with $\mu(t)$. From this, the mean value function of the fault correction process is expressed as:

$$\frac{dm_c(t)}{dt} = \mu(t)(m_d(t) - m_c(t)) \tag{4}$$

It can be shown that using Eq. (4) to describe the fault correction process, and defining $C(t) = \int_0^t \mu(s)ds$ (i.e., the cumulative correction rate per fault), the cumulative number of corrected faults is given by Lo and Huang (2006):

$$m_c(t) = e^{-C(t)}\left(\int_0^t ac(s)e^{C(s)}m_d(s)ds\right) \tag{5}$$

where $m_c(t)$ represents the cumulative number of detected and *corrected* faults, and $C(t) = \int_0^t \mu(s)ds$ models the FCP by means of the *fault correction rate per detected but not corrected fault,* μ. Depending on the SRGM chosen, this Equation can also model the *imperfect* debugging: there is a class of SRGMs known as infinite-failure models, which, contrarily to the finite-failure models, assume that an infinite number of faults would be detected in infinite testing time (Lyu, 1996). These are meant to capture the case where faults may be reintroduced during debugging: an example is the Musa-Okumoto logarithmic Poisson execution time model (Musa and Okumoto, 1984) and the more recent failure-size proportional model proposed in Zachariah and Rattihalli (2007). While many studies refer to the fault detection process only, there are TRAP studies considering the fault correction process too (hence, using *debug-aware* SRGMs), as will be shown in the following Sections.

Finally, it should be noted that SRGMs can be used in various ways to express the reliability of the software in a TRAP. The conventional way is to consider the expected number of detected (or corrected) faults $m(t)$ for each module, in either the objective or constraint. However, using a function of $m(t)$ makes sense, such as the mentioned *fault detection rate per remaining fault* λ (i.e., failure intensity) or the expected reliability at operational time.[5]

[5] Reliability at operational time t is: $R(t) = \exp(-\lambda(T)\cdot t)$, where $\lambda(T)$ is the failure intensity at the end of testing (at time T) (assuming no change in the software during operation (Pietrantuono et al., 2010))

Table 1
Numerical Example of testing resources allocation.

Component	GO SRGM Parameters		Optimal Allocation		Size (KLoC)	Size-based Allocation	
	a	*b*	Effort	Exp. # of det. faults		Effort	Exp. # of det. faults
C_1	120	9.41E-2	18.95	99.83	39	18.39	98.74
C_2	98	5.11E-2	18.98	60.86	55	25.94	71.96
C_3	102	2.34E-2	9.79	20.89	22	10.37	21.99
C_4	194	2.89E-2	37.56	128.98	59	27.83	98.05
C_5	78	8.52E-2	14.72	55.72	37	17.45	60.36
Sum	**592**	–	**100**	**366.29**	–	**100**	**351.13**

As for the third objective, the *cost*, several models are available (Yang et al., 2015a). In any case, the goal in a TRAP is to minimize the sum of costs spent to test each module — hence the total testing (and, possibly, debugging) cost. A common model is the following one (Huang and Lo, 2006a), (Yamada et al., 1993):

$$\begin{aligned} C(t) &= C_1^* \cdot (\delta/24) \cdot m_c(t) \\ &+ C_2^* \cdot (\delta/24) \cdot (m_d(\infty) - m_c(t)) \\ &+ C_3^* \cdot (Y/24) \end{aligned} \tag{6}$$

where:

- C_1^* is the cost per person-day to correct a fault during testing;
- C_2^* is the cost per person-day to correct a fault at runtime (typically $C_2^* \geq C_1^*$ (Boehm, 1981));
- C_3^* is the cost per testing-effort expenditure unit (e.g., person-day), i.e., hourly or daily cost of a tester;
- δ is the average number of hours to fix a fault.

Other models encountered in a TRAP include variants of Eq. 6, for instance neglecting the debugging cost, or the simpler exponential model: $C_i = k_1 exp[k_2 R_i - k_3]$, with R denoting reliability of component i, and k_1, k_2, k_3 are model's parameters inferred by data (Yang et al., 2015a).

2.3. Example

In the following, an example of a testing resource allocation problem is illustrated in order to show how it can be formulated and what are the expected benefit of applying a TRAP approach. To be concrete, let us consider the case of a real *homeland security system* used in one of the surveyed studies (Carrozza et al., 2014). The system is a large software-intensive application made up of 5 components, C_1 to C_5. Each of such components is an independent unit called *Computer Software Configuration Items* (CSCI): a CSCI is actually a large deployable component – their size in the mentioned case study ranges from 39 KLoC to 59 KLoC – that is developed, tested and maintained independently. Assume to have a budget of $T^* = 100$ units of testing effort (e.g., man-days) to distribute between such components, with testing effort being linearly related with testing time.[6] Let us further assume that failure data from previous releases of the components are available (e.g., from previous testing sessions or from operation) and that the objective of testing is to maximize the number of detected faults before release, given the budget. Consider, for simplicity, the Goel-Okumoto (GO) exponential SRGM to fit the failure data for all the five components. The GO model's mean value function (*mvf*) is: $m_d(T) = a(1 - e^{-bT})$, where a is the expected number of total faults, b is the (constant) fault detection rate per remaining fault and T is testing effort. Columns 2 and 3 of Table 1 report hypothetical values for the parameters of the SRGMs.

Under these assumptions, the total expected number of detected faults is the sum of the *mvf*s of the five components (let us denote them as $m_{d_i}(T_i)$), each evaluated at the *effort* value T_i suggested by the allocation problem's solution. The optimization problem of this example looks like this:

$$max \quad m_{d_s} = \sum_{i=1}^{5} m_{d_i}(T_i) \qquad s.t. \qquad T = \sum_{i=1}^{5} T_i \leq T^* \tag{7}$$

where m_{d_s} is the total expected number of faults detected. Solving this problem gives the optimal allocation reported in Table 1, yielding an expected total number of detected faults equal to 366.29 (out of 592 expected faults detectable with an ideally infinite testing effort). The solution tends to give more resources to components for which the expected fault detection rate is bigger. If we consider the size of each component to allocate testing resources, which is an intuitive criterion to distribute testing resources, then results are in the last three columns: a proportional-to-size allocation yields an expected number of faults equal to 351.13. A further hypothetical case of a uniform allocation (namely, 20 man-days for testing each component) provides an expected number of detected faults equal to 343.42.

The example shows the benefits expected from applying an optimal TRAP solution compared to other, less accurate, criteria. The same applies whenever the objective is to minimize the testing effort or testing cost, given a minimum level or reliability to attain.

3. Study design

To explore the TRAP research area, well-established guidelines for systematic mapping (SM) studies are followed (Petersen et al., 2015; Kitchenham and Brereton, 2013). The high-level objectives are those typical of SMs, namely *i)* to examine the extent, range and nature of the research activity, *ii)* to summarize and disseminate the main research findings, and *iii)* to identify research gaps in the existing literature (Arksey and O'Malley, 2005).

3.1. Research questions

The study targets the following research questions:

RQ1 — *What are the publication trends of research studies about TRAP?* The goal of this RQ is to characterize the intensity of scientific interest for this research area, the base of researchers working on it, the relevant venues where they publish their results, the type of research they conduct.

RQ2 — *How are TRAP models formulated?* This RQ aims at characterizing how researchers define objectives and constraints in the TRAP models, how the relation between component-level and system-level attributes is considered, and how the models should be applied.

[6] If testing effort is assumed to grow linearly with testing time, then the SRGM can indistinguishably use effort or time measure as independent variable, namely there is no need for a so-called *testing effort function* (TEF) to account for the non-linear relation between time and effort.

RQ3 — *How are TRAP models solved?* This RQ characterizes the type of solutions chosen by researchers and which are the solutions that generally perform better.

RQ4 — *How are TRAP models evaluated/validated?* This RQ looks at how TRAP models and solutions are evaluated or validated, so as to provide figures on the extent to which TRAP models can be applied in practice. In order to answer these RQs, a classification of TRAP studies is defined, which will be useful to analyze the current trends and to systematically frame future research.

3.2. Study identification process

The following selection process was followed:

1. **Initial search and filtering**. The initial selection was performed by a keywords-based search on three major digital libraries in computer engineering and computer science — *SciVerse Scopus, IEEE Xplore, ACM Digital Library. IEEE Xplore* and *ACM Digital Library* are publisher-specific databases widely used in empirical software engineering studies. I selected Scopus DB as generalist indexing DB, which is the largest indexing DB for peer-reviewed literature, to complement the results with a wide range of other publishers, such as Springer, Wiley, Taylor and Francis, Sage, Hindawii, McMillan and others. Key researchers in empirical software engineering, like Petersen and Kitchenham, also use Scopus for their studies, e.g., (Petersen et al., 2015; Kitchenham and Brereton, 2013).
   The search string was kept generic so as to cover as much relevant studies as possible: either the title, the abstract or the keywords of a paper were required to include the following keywords: *(Software AND Testing AND Resource* AND Allocation)*. This provided 542, 981 and 157 studies from the three libraries, respectively, including duplicates. Due to the conservative search, the vast majority of the initial set were found to be unrelated to the computer engineering/science — either belonging to other engineering fields, such as materials, civil or electrical engineering, or even to non-engineering research areas, such as business management, decision sciences, manufacturing. Also, some results were not research articles, but editorials, standards, conference proceedings material (e.g., welcome messages, ToC). Such irrelevant results were removed based on the papers' title and publication venue, getting to an initial set of 265 studies from the three libraries, with duplicates being removed.
2. **Application of selection criteria**. On the set of 265 studies, the following inclusion/exclusion criteria were applied:
   - *Inclusion Criterion 1*. Studies targeting the software testing resource allocation problem.
   - *Inclusion Criterion 2*. Studies subject to peer review.
   - *Inclusion Criterion 3*. Studies written in English.
   - *Exclusion Criterion 1*. Studies proposing solutions that could potentially support the testing resource allocation, but in which the allocation is not the primary focus, such as studies on: software reliability growth models (SRGM), fault/defect prediction, software change prediction, risk/quality assessment, reliability assessment.
   - *Exclusion Criterion 2*. Studies focusing on allocation strategies of resources unrelated/not applied to *testing* (i.e., just *potentially* applicable to testing resources), such as allocation of cloud resources to end users.
   - *Exclusion Criterion 3*. Studies focusing on testing techniques (e.g., test prioritization, test selection) that could be viewed as strategies to "allocate" tests to partitions, but whose focus is not on the testing resource allocation problem.
   - *Exclusion Criterion 4*. Secondary or tertiary studies (e.g., systematic literature reviews, surveys, etc.).
   - *Exclusion Criterion 5*. Studies not available as full-text.

   Applying these criteria, the initial set of 265 papers is reduced to 60 papers.
3. **Snowballing**. The process is completed by considering each paper in the initial set and adding relevant papers either citing and cited by it. The set after snowballing is of N=65 studies.
4. **Evaluation**. I have used the test-set method for validating the study identification process (namely, for search evaluation as well as for inclusion/exclusion): I looked for a test-set of known papers that ought to be found and checked if the search found them — this is the most common strategy. In particular, I created two validation sets for double checking the search. The first one is obtained by selecting key researchers in the field of testing resource allocation. As suggested by Petersen et al. (2015), I looked at their profile to find papers related to the TRAP, and created a list of 7 papers (10% of the sample) that should indeed be in my search (i.e.: Ohtera and Yamada, 1990; Yamada et al., 1995; Lo et al., 2002; Lyu et al., 2002; Huang and Lyu, 2005b; Huang and Lo, 2006b; Yang et al., 2015a). From this validation set, it turned out that they are all included in the search. This set is also used to double-check the inclusion/exclusion criteria objectivity (Petersen et al., 2015), and resulted in the addition of one inclusion criterion and one exclusion criterion:
   - *Inclusion Criterion 4*. Studies dealing with the resource allocation based on the operational profile or on requirements should be kept, provided that the resources are distributed to components/modules implementing those operations/requirements (namely: the method does not allocate resources *to* requirements/operations, but it allocates resources to components *based on* requirements/operations).
   - *Exclusion Criterion 6*. Studies dealing with the resource allocation to testing phases (i.e., how much testing for unit/integration/system testing) or to testing techniques should be excluded.

   This revision led to the exclusion of 2 papers.
   To complement the first set, the second validation set is created by selecting three reference papers with the highest citation ratio (number of citations over number of years since their publication) (Lyu et al., 2002), (Huang and Lyu, 2005b) and (Huang and Lo, 2006b), and then considering all the citing papers — namely by a forward snowballing step. This turned out in a list, with duplicates removed, of 113 papers. After applying inclusion/exclusion criteria, the set was reduced to 27 papers. From the second set it turned out that 5 papers had been missed, and have been added to the list. Thus, the final set is N=65-2+5 = 68 papers. The list is available at: https://github.com/rpietrantuono/IS2018.
5. **Quality assessment**. Quality assessment is not a mandatory step in systematic mapping, unlike systematic literature reviews, but it is deemed useful to enforce the procedure robustness, e.g., to assure that sufficient information is available to be extracted (Petersen et al., 2015). The quality assessment should not pose high requirements on the primary studies, as the goal of a mapping study is to give a broad overview of the topic area (Kitchenham et al., 2010). Thus, I have adapted the approach used by Petersen (Petersen et al., 2015), and asked the following questions, each with three sub-questions:

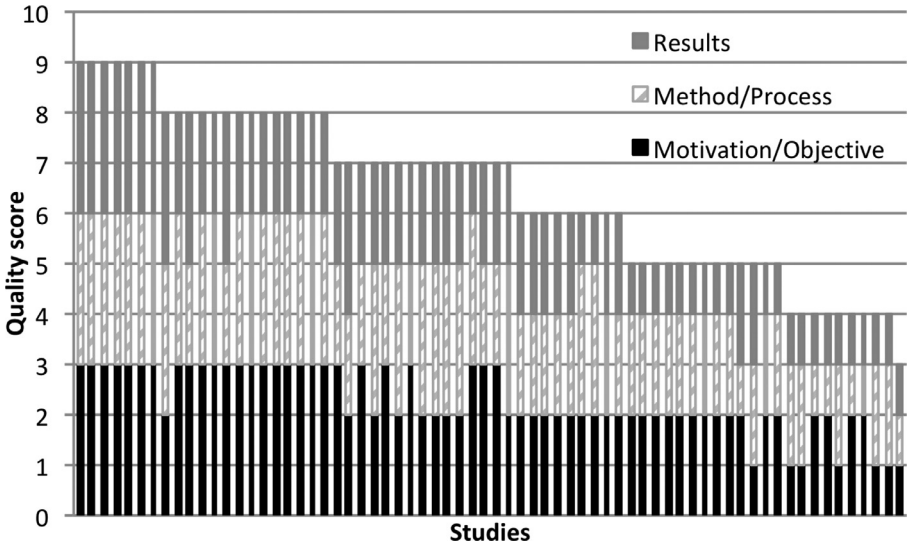

Fig. 1. Results of quality assessment.

- **Motivation/Objective**: *1) Is the motivation for a TRAP solution proposal clearly stated? 2) Is there a clear statement of the aim of the paper? 3) Is there an adequate description of the primary outcome?*
- **Method/Process**: *1) Is the TRAP formulation clear? 2) Is the TRAP solution clearly described? 3) Is the TRAP validation process reported?*
- **Results**: *1) Is there a clear statement of the findings? 2) Are the limitations of the study discussed explicitly? 3) Are the results consistent with (and adequate to support) the claims?*

A score [0,3] is assigned to each of the three aspects based on the satisfaction of the corresponding sub-questions — hence for a maximum score of 9. The scores are reported in Fig. 1. No paper is excluded due to low quality. The main flaw was in the "result" criterion, which is often poor in terms of support to the claims and limitations discussion — a point further highlighted later in the paper. Other aspects are discussed in subsequent analyses.

3.3. Data extraction and classification

To extract data from the identified primary studies, I used the template shown in Table 2. Each data extraction field has a data item and a value. The extraction process is validated using the same strategy adopted for inclusion/exclusion criteria, namely by the test-set approach on the list of the seven reference papers mentioned in the previous Section, and by checking whether extracted data is enough to cover all the relevant aspects targeted in these pilot papers (Petersen et al., 2015). No revision of the extraction form has been needed after the check on the pilot.

To analyze the studies, a classification scheme is defined based on four main dimensions related to the four research questions, each with several attributes of interest. Starting from the extraction template, the scheme has been iteratively refined based on the revision of each paper and the extracted data. The identified papers have been then examined with respect to the classification scheme, summarized in Fig. 2.

Publication trends. The first dimension refers to the publication trends and includes these attributes: number of publications by year, publication type (i.e.: journal, conference or workshop) and venue, publications' authors, and type of contribution (i.e., research type). The latter attribute is based on a common scheme from Wieringa *et al.* (Wieringa et al., 2006), which distinguishes papers as in Table 3, under the *Decision* row. The Table also reports the conditions, in terms of rules, to be satisfied for a paper to be classified in a research type category (Petersen et al., 2015).

Formulation. The second dimension refers to several attributes about how the TRAP is formulated. It distinguishes the type of optimization (single- vs multi-objective); the optimization goals and constraints (i.e. in terms of: reliability, testing cost, testing time/effort) — for each goal several attributes are considered about the type of model being adopted; the way in which the system architecture is modeled; the allocation strategy (static or dynamic).

Solution. The third dimension refers to how the TRAP is solved, namely if an exact method is adopted rather than a heuristic or metaheuristic approach. Beside the method, we look at which specific algorithm is adopted, along with the result of possible algorithm's comparisons performed in each study (with related metrics). Exact methods refer to the algorithms to solve non-linear programming problems (NLPP), which are the typical way to represent the resource allocation optimization problem. Examples are the algorithms based on Lagrangian multipliers or dynamic programming approaches. Such algorithms allow finding optimal solutions, but are often extremely time-consuming when solving real-world problems (e.g., with large dimensions). Heuristic and metaheuristic techniques are powerful and flexible search methodologies that have successfully tackled practical difficult problems. The algorithms do not guarantee the optimal solution, but they attempt to produce good-enough solutions for practical purpose in reasonable computation times, and can tackle efficiently multi-objecitve and many-objective cases. While heuristic algorithms are specific and problem-dependent, a metaheuristic is a high-level problem-independent framework that provides a strategy to develop heuristic algorithms. Common examples are genetic algorithms, simulated annealing or tabu search.

Evaluation/Validation. The fourth dimension is about how the TRAP solution is evaluated/validated. The first attribute is *research method*. Wieringa et al. (Wieringa et al., 2006) distinguishes the following research methodologies frequently applied in software engineering: survey, case study, controlled experiment, action research, ethnography, simulation, prototyping, and mathematical analysis. These apply to *evaluation research* and *validation research*, namely wherein the *empirical evaluation* condition applies (Table 3), and need to be consistent with these two categories. I exclude survey and ethnographic, as they are incompatible with

Table 2
Template for data extraction.

Data Item	Value/Description	RQ
General		
Study ID	Integer	
Title	Name of the article	
Authors	Name of the authors	RQ1
Year	Year of publication	RQ1
Venue	Venue of publication	RQ1
Citations	Number of citations	
Research type	What type of research is conducted (based on Wieringa scheme (Wieringa et al., 2006))	RQ1
Research method	What research method is used (based on Wieringa scheme (Wieringa et al., 2006))	RQ4
TRAP-related		
TRAP objectives	What objective functions are used	RQ2
TRAP models	What models are used for the formulation	RQ2
TRAP constraints	What are the constraints of the TRAP	RQ2
Architecture	How the architecture is considered	RQ2
Fault tolerance	How fault tolerance is considered	RQ2
TRAP solution	What solution strategy are used	RQ3
TRAP algorithms	What solution algorithms are used	RQ3
Eval/Val: Industry-related	Industrial involvement in TRAP evaluation/validation	RQ4 RQ4
Eval/Val: Problem size	Size in terms of modules/components	RQ4
Eval/Val: Dataset	What dataset used is real or not	RQ4
Eval/Val: Tools	What, if any, tool is used to solve the TRAP	RQ4
Eval/Val: Sensitivity analysis	If a sensitivity analysis is carried out or not	RQ4

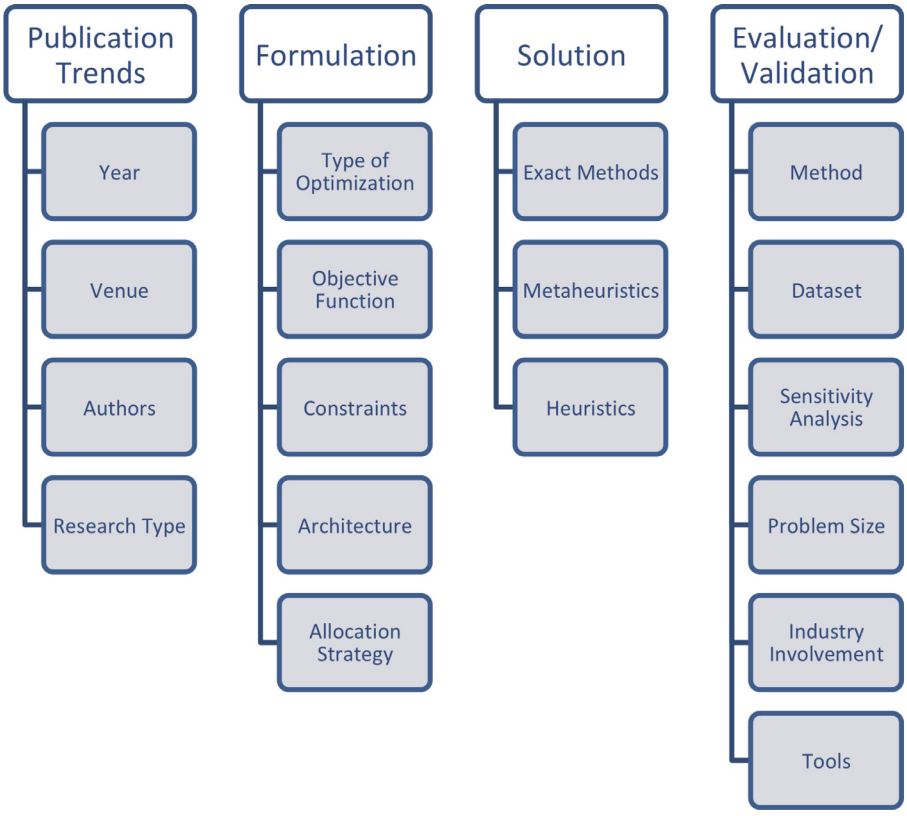

Fig. 2. Classification of TRAP studies.

the used exclusion criteria, ending up with the scheme exemplified in Fig. 3, adapted from Petersen (Petersen et al., 2015). Other attributes of this dimension include: if the dataset is real or not, if industry is involved, if a tool (already existing or developed ad hoc) is used for solution, if sensitivity analysis is run and, finally, what is the size of the problem being addressed.

4. Results

4.1. Publication trends

Publications by year. Fig. 4 plots the selected studies by year and publication type (*Journal, Conference* or *Workshops* proceedings). Papers range from 1987 to 2017 (the last year considered

Table 3
Research types facets (Wieringa et al., 2006). The • sign means "irrelevant".

	R1	R2	R3	R4	R5	R6
Conditions						
Used in practice	T	•	T	F	F	F
Novel solution	•	T	F	•	F	F
Empirical evaluation	T	F	F	T	F	F
Conceptual framework	•	•	•	•	T	F
Opinion about something	F	F	F	F	F	T
Authors' experience	•	•	T	•	F	F
Decision						
Evaluation research	√	•	•	•	•	•
Proposal of solution	•	√	•	•	•	•
Validation research	•	•	•	√	•	•
Philosophical papers	•	•	•	•	√	•
Opinion papers	•	•	•	•	•	√
Experience report	•	•	√	•	•	•

Table 4
Average quality score by venue type.

Venue type	Motivation/objective	Method/process	Results	Total
Journal	2.55	2.525	2.025	7.1
Conference	1.923	2.115	1.5	5.54
Workshop	2	1.5	2	5.5

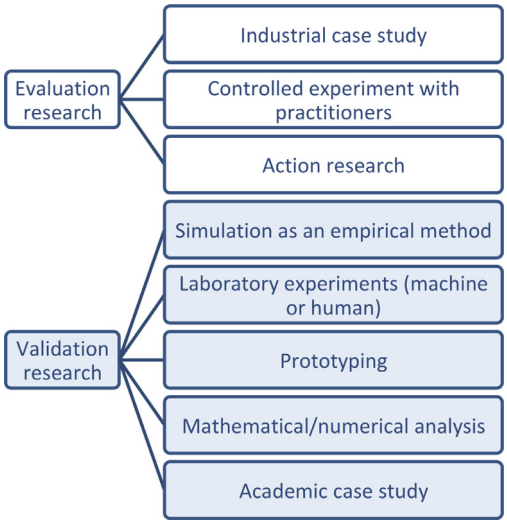

Fig. 3. Classification of research methods (modified from Petersen et al. (2015)).

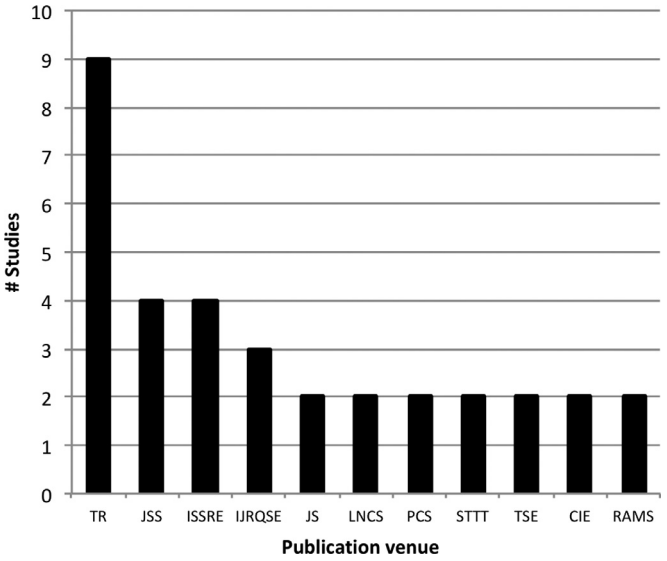

Fig. 5. TRAP studies by venues.

in our search, including *early access* papers available online). However, a substantial activity on TRAP started in 2002. Since then, 3.5 papers per year appeared in the literature, with more papers in the last 5 years (22 studies in 2013–2017 compared to 17 studies in 2008–2012 and 18 studies in 2003–2007). Most of considered studies are in journal papers (42/68), followed by conference proceedings (24/68) and very few workshop papers (2/68). Table 4 reports the quality score by venue type (journal, conference, workshop); journal articles have been generally assessed as higher-quality studies.

Publications venues. Fig. 5 details the main publication venues. In half of the cases (32/68), a journal/conference hosted more than study. The most targeted journal is IEEE Transactions on Reliability (TR), with 9/68 occurrences, followed by the Journal of Systems and Software (JSS) with quite fewer studies (4/68). The main targets are reliability-related venues, such as TR (9/68) the International Symposium on Software Reliability Engineering (ISSRE) (4/68); the International Journal of Reliability, Quality and Safety Engineering (IJRQSE) (3/68), Annual Reliability and Maintainability Symposium (RAMS) (2/68), and software-engineering-related venues, such as JSS (4/68), Journal of Software (JS) (2/68), IEEE Transactions on Software Engineering (TSE) (2/68), Procedia Computer Science (2/68), International Journal on Software Tools

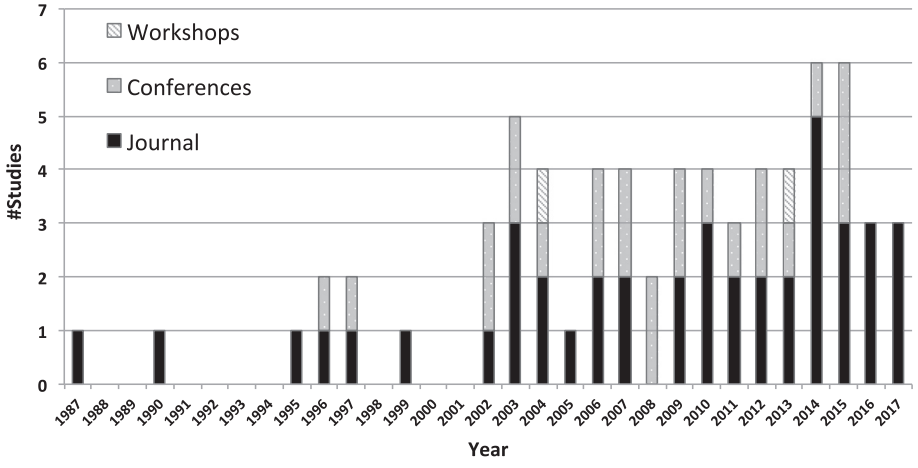

Fig. 4. TRAP studies per year and venue type.

for Technology Transfer (STTT) (2/68), Lecture Notes in Computer Science (LNCS) (2/68). In the remaining 36/68 cases, the journal/conference hosted just 1 TRAP paper. Overall, the 68 studies were published in 46 different venues, covering quite a large range of journals and conferences.

Research groups. The total number of different authors for the 68 studies is 120. Authors who co-authored more than 1 paper accounts for exactly 1/3 of the total (40/120), while just a few of these (19/120) published more than 2 papers. The remaining 2/3 (80/120) published exactly 1 study on TRAP. Some authors (with their research group) are extremely active on the topic, with more than 4 (8/116), 5 (4/120) or even 7 (2/120) published studies. It is worth to mention the name of researchers who published more: *Huang C.-Y., Kapur P.K., Lo J.-H., Lyu M.R., Khan M.G.M., Ahmad N., Kuo S.-Y.* are the researchers who published, respectively, 8, 7, 6, 6 4, 4, and 4 studies on TRAP.

Research type. Among the six types of research outlined in Table 3, *validation* and *evaluation* research are those adopted by TRAP researchers. In fact, all the studies report a form, more or less detailed, of empirical evaluation for their method. In the vast majority of the cases (61/68), there is an empirical validation but the technique is not used in practice (*validation* research); in seven cases, researchers conducted an *evaluation* research, in which, beside an empirical evaluation, they implement and apply the method in practice, for instance through industrial case studies or action research. The specific method used for evaluation/validation is analyzed in the corresponding dimension (RQ4) – results are in Section 4.4. No study reports just about the authors' experience in practice (namely, *experience report*), and no study proposes a novel solution without an empirical evaluation (namely, *proposal of solution* category); there are no philosophical and opinion studies.

The following findings summarize the main results.

F1 *The TRAP topic is studied constantly since 2002, with an increase in the last 5 years. Most of papers are in journals (and often in top journals, such TR or JSS), which, generally, have been attributed a higher quality score than conferences/workshops Hence, the field is well-rooted in practice (with an increasing interest likely due to new challenges posed by modern development and testing paradigms) and mature enough to produce mostly journal-level contributions.*

F2 *Looking at where researchers tend to publish, the TRAP topic matches the interest of a large variety of communities in software reliability and software engineering areas.*[7] *Studies are targeted equally to reliability (21/68) and to software engineering/computer science (23/68) communities (others are published in broader communities, e.g., computing technologies, math and operations research areas (24/68)). Those published in reliability-related venues are less scattered, focused on fewer journals/conferences.*

F3 *The TRAP area involves quite a high number of researchers (120 for 68 studies). There is a high diversity in terms of people working on the topic (2/3 of researchers published only 1 paper). 8/120 are the researchers (with their research groups) that worked and are working more actively on the topic.*

F4 *All the studies report some form of assessment for their method, but just seven studies evaluate their method in practice, e.g., through industrial case studies or action research. The remaining 61/68 studies are validation research.*

[7] It should be noted that the distinction between reliability and software engineering/computer science communities does not refer to the authors, who can hardly be attributed to one or another community, but solely to the venue of publication. The latter is based on the venue's source title (and, in ambiguous case, on the "aim and scope" of the venue), distinguishing the cases where the main (not the sole) focus is clearly on one of the two areas, while judging as "other" the overlapping cases.

Table 5
TRAP studies by *Formulation*.

Formulation	Frequency
Type of optimization	*n* out of 68
Single-objective	60
Multi-objective	8
Objective Functions*	
Reliability (R)	42
Time/Effort (T/E)	25
Cost (C)	24
Others (O)	4
Constraints*	
Reliability (R)	34
Time (T/E)	58
Cost (C)	4
Others (O)	–
Architecture Model†	
Usage/Weight factor	19
Parallel-series model	7
Markovian model	5
Others	4
None	36
Allocation Strategy	
Static	56
Dynamic	12

* *The sum of occurrences is greater than 68, as MO studies consider more of them together*
† *The sum of occurrences is 71, as 2 studies considers more architectural models together*

4.2. Formulation

Table 5 reports the results with respect to the *formulation* dimension and its attributes.

Type of optimization. The first part of Table 5 reports the number of studies formulating the TRAP as single or multi-objective problem. Clearly, most of research is on single-objective optimization, while multi-objective is a more recent research topics — all of them appeared after 2008 and 6/8 appeared after 2013.

Objective functions and constraints. Most of the studies consider the maximization of reliability as objective, followed by the minimization of time/effort or cost of testing. Specifically, reliability is a primary concern (in **42/68** studies) in both single-objective (SO) formulations (34/60 SO studies) and in multi-objective (MO) formulations (8/8 MO studies). It is also quite often set as a constraint in the optimization model as minimum level of reliability to assure, with testing time/effort or cost as objective to minimize — especially in SO formulations (31/60 SO and 1/8 MO studies).

To express reliability, an SRGM is used in **47/68** studies and in total 55 times (considering studies that explore more than one SRGMs). Most of the times the exponential GO model (one of the earliest and simplest one) is adopted (29/55), followed by the S-shaped model (8/55). Other models employed in a TRAP formulation include Weibull, log-logistic, log-normal, hypergeometric model, Schneidewind model (5/55). In few other cases, the expression of the SRGM is not specified, namely the optimization model is supposed to work under a generic SRGM (10/55). Researchers also integrate *debug-aware* SRGMs (see Eq. (5)) into a TRAP formulation, but more rarely (9/68), adopting a fault correction rate

(μ) of either exponential form (2/9), S-shaped (1/9), or unspecified (3/9).

Testing time or effort is the second most used objective (**25/68**). A time/effort objective is used in 19/60 SO studies and, together with other objectives, in 6/8 MO studies. *Testing effort* is more common (17/25) than testing time (8/25). Effort is in almost all the cases (50/68) set as constraint, since it represents the available budget that should not be overcome during testing. Testing time is set as constraint in 8/68 cases to represent the project schedule deadlines.

In **26/68** studies, models account for the relation between the *testing effort* (in terms of person-power) and *testing time* (in terms of calendar time, CPU time or number of test cases). In fact, while the simpler case assumes the testing effort E varying linearly with time T, this is, in general, not true. In the literature, the relation is modeled by the so-called *Testing Effort Functions* (TEFs), which model such a non-linearity by a function F: $E = F(T)$. The most common TEF, shown to well represent the usual trend of testing effort, is the logistic TEF (Huang and Lo, 2006a; Huang et al., 2007; Huang and Lyu, 2005a), given by the following equation:

$$Y(t) = \frac{\mathcal{B}}{\sqrt[h]{1 + \mathcal{A}exp[-\alpha ht]}} \tag{8}$$

where $\mathcal{B}$ is the total testing effort to be consumed; α is the consumption rate of testing-effort expenditures; h is a structuring index depending on the software development process – a bigger value models well-structured processes and the spent effort converges sooner to the budget $\mathcal{B}$ (the parameter can account for improvements in the software development process, such as stepwise refinement or top-down approach); $\mathcal{A}$ is a constant, whose value regulates the testing effort spent at the beginning and the "distance" from the budget $\mathcal{B}$ (the bigger it is, the less testing effort is spent at the beginning of the process); and $y(t) = \frac{dY(t)}{dt}$. When considering a TEF, $m(t)$ in Eqs. (3) and (5) changes, since the effort Y is considered in lieu of time, making it more complex to solve. Logistic (5/26), Weibull (5/26), log-logistic (2/26) and exponential (3/26) are some of the used TEFs.

Finally, a *cost* function as minimization objective is found in **24/68** studies, in more recent works. All the MO models have cost as objective (8/8), while it is less common in SO models (16/68), especially the oldest ones. Cost is a measure related to the effort spent, but goes beyond it. The model in Eq. (6) is the most used one (9/24), followed by simpler exponential cost models (6/24). In only 4/68 cases (all SO cases), cost is used as constraint (i.e., a monetary upper bound budget); most often, effort is used in lieu of cost as limiting budget.

The *other* category includes four cases in which a measure of *risk* is employed to represent an objective to minimize under what is known as risk-based testing (Felderer and Ramler, 2014a).

Architecture modelling. Several formulations consider how the relation between modules impacts the resources allocation. For instance, a reliability maximization objective cannot deal only with *how many* faults are in the software, but also with *how often* they could be activated at runtime. A software module with more faults than another can be more reliable inasmuch as those faults are less often activated and cause less frequent failures in operation (Cotroneo et al., 2013). Thus, if the system's reliability computation does not account for how of10 a module is used, the solution could assign a lot of testing resources to a rarely used module, which will not actually contribute to a reliability improvement, wasting resources.

More than half of the studies (**36/68**) do not account for any form of inter-module relation. In the other cases, the most adopted solution is to consider a usage factor (19/68), namely a [0–1] weight expressing the frequency (or probability) with which a module will be involved. An alternative is to consider structures of parallel-series (or more complex ones, like bridged, star) and the associated structure function, as in Reliability-Redundancy Allocation Problems (Kuo and Wan, 2007; Zhang et al., 2017). In TRAPs, this was adopted in 7/68 cases. A more complete solution is to consider Markovian architectural models, like Discrete-time Markov Chains (DTMC), representing modules by states, with transition probabilities describing the execution flow from one component to another (Pietrantuono et al., 2010; Yang et al., 2015a; Fiondella and Gokhale, 2012). The derived measure, the so-called *visit count* (i.e., the average number of visits to a component), is used in the system-level reliability computation. This is adopted 5/68 times. Other solutions (4/68) include: relying on *expert judgment* to spot the most critical modules; exploiting design-time documentation (UML use cases); assuming an equal importance of modules.

A further factor impacting on the architecture is the inclusion of fault tolerance means. A failure in a module may be tolerated, thus keeping the system up despite the failure. Therefore, fault tolerance improves the reliability of a module, impacting on the overall reliability computation. Redundancy is one of such means. However, in TRAP literature, only 3/68 studies considered fault tolerance means in their formulation. Two of these include a [0,1] *coverage* factor in the module-system architectural relation aimed at decreasing the failure probability of a fault-tolerant module compared to a non-fault-tolerant one (Lyu et al., 2002; 1997). The remaining one adopts a more complex solution in which the reliability of a module is computed according to possible fault tolerance means it implements in case of failure, such as restart, retry or failover (Pietrantuono et al., 2010).

Allocation strategy. A problem with the usage of models (especially the SRGMs) are the assumptions they rely on, which, if strongly violated, can lead to inaccurate modelling (e.g., inappropriate choice of SRGM and/or large uncertainty in the parameters) and a wrong optimization. This is exacerbated in large systems, where an accurate upfront planning is difficult because of the high variability from the planning time to the testing completion. Testing a single module may in fact require months; during this period, changes related to the testing process, the environment, the personnel, and the technology can affect the result in terms of defect detection, and are likely to invalidate any assumption made at planning time. Moreover, the diversity of tested units, of their development teams, and of testing teams as well, is a further factor of variability. Under these circumstances, the "best" SRGM for each module cannot be established a priori. Such contexts call for a *dynamic* support to test planning, to be robust to unplanned variations. Though, dynamic allocation is used in a minority of the surveyed studies (**12/68**). Moreover, the literature is still stuck on parametric models, e.g., to represent the reliability growth process, which are known to make more assumptions and to be more sensitive to them. The main findings about the formulation of TRAP models are reported hereafter.

> **F5** *Reliability is the most common objective function in a TRAP model, followed by testing effort/time. Although effort is an indirect measure of cost per se, an explicit cost function (i.e., considering testing effort expenditure along with development-time and/or operational-time debugging cost) is used quite often (24/68). In all MO models (8/68), thus in most recent papers, a cost function is present in all the cases. Considering the union of studies considering effort or cost (i.e., "resources" expenditure) as objective, 17+24 = 41 studies have a cost-related objective, just like reliability. Time is the least used objective function.*

> **F6** *Testing time/effort is almost always present as a constraint — hence the models represent the most common situation where a given budget is available and should not be overcome. The use of TEF to model the time-effort relation is well established, as a TEF*

is present in 26 cases despite the further complexity required to handle it.

F7 *Despite the practical importance to obtain results close to the reality, the architecture is considered in only half of the studies (35/68). When considered, a simple usage factor is included, while more complex relations are left unaddressed. Similarly, fault tolerance, which is the core aspect in R-RAP models, is mostly neglected in TRAP, wherein the only means to improve component's reliability is assumed to be testing.*

F8 *A dynamic allocation approach, aimed at having solutions more robust to the possible violation of assumptions, is receiving attention by some researchers (12/68). The formulation of models that are alternative to parametric ones, with less assumptions, is not explored yet by the TRAP community.*

4.3. Model solution

Table 7 reports the results with respect to the *solution* dimension and its attributes.

Exact Method. The optimization models formulated as above are non-linear programming problem (NLPP). Many studies adopt an exact method to solve it (**32/68**) — in all the cases, the problem is a *single-objective* NLPP. The most popular methods are by far those based on *Lagrange multipliers* (20/32), customized to accomodate specific problem formulations. Relevant examples include the works by Lyu et al. (Lyu et al., 2002; 1997; Lo et al., 2002), and by Huang et al. (Huang and Lo, 2006b; Huang and Lyu, 2005b; Huang et al., 2004).

Dynamic programming is a valuable alternative (5/32). For instance, in Khan et al. (2016), the authors exploit dynamic programming to solve two allocation problems, aiming at *i)* maximization of the number of faults removed when the amount of testing-effort is fixed, and *ii)* maximization of the number of faults removed satisfying a certain percentage of initial faults to be removed with a fixed amount of testing-effort.

Other methods include conventional techniques to solve NLPP, such as the generalized reduced gradient (GRG) and sequential quadratic programming (SQ) (both 3/32); the gradient projection (GP) and sequntial linear programming (SLP) (both 1/32). In most cases, the solution is analytically determined without reference to a specific support tool for solution. In few cases (5/32), a specific tool is mentioned: LINGO (2/32), Matlab (2/32), Excel (1/32). When exact methods are used, researchers do not focus on comparing the chosen solution method with others (it happens just in 2/32 cases), e.g., in terms of solution time and scalability. The evaluation focuses solely on showing how the TRAP solution provides the expected results.

Metaheuristic. While traditional optimal TRAP advocates the usage of exact methods, metaheuristic approaches are gaining popularity. For complex system configurations and/or with multiple contrasting objectives to optimize, exact methods would not scale well. Thus, an increasing number of studies is relying on metaheuristics (**18/68**). Our previous study explore the usage of metaheuristics for TRAP (Pietrantuono and Russo, 2018) — here the main results are reported.

Out of 18 studies, 10 deal with SO problems and 8 with MO problems (i.e., all the 8 MO studies use metaheuristcs). As shown in Table 6, the most used metaheuristics are (variants of) the NSGA-II (Non-dominated Sorting Genetic Algorithm) followed by the HaD (Harmonic Distance) algorithm for MO problems. For SO problems, the GA is by far the most used technique.

Other popular algorithms for the MO cases are those based on Differential Evolution, namely Multi-Objective Differential Evolution (MODE), Weighted Normalized Sum-Multi-Objective Differential Evolution (WNS-MODE) and the hybrid Cellular Differential Evolution (Cell-DE), combining Multi-objective Cellular (MOCell) and MODE, which account for 5 cases altogether. Well-known older metaheuristics are also tried for TRAP, such as: Pareto Archived Evolution Strategy (PAES), Multi-objective Particle Swarm Optimization (SMPSO), and Indicator-based Evolutionary Algorithm (IBEA). For the SO case, other algorithms, beside the conventional GA, are: a variant of GA, the Genetic Local Search Algorithm (GLSA), combining GA with local search; the well-known Hill Climbing (HC) and Simulated Annealing (SA).

Table 6
TRAP studies by *Solution*.

Solution	Frequency
Exact methods	32 out of 68
Lagrange multiplier	20
Dynamic programming	5
Generalized reduced gradient	3
Sequential quadratic programming	3
Gradient projection	1
Sequential linear programming	1
Metaheuristics	18 out of 68
Mulit-objective*	
NSGA-II	7
HaD	4
MODE	2
WNS-MODE	2
PAES	2
MoCELL	2
Others	6
Single-objective*	
GA	6
GLSA	1
Hill Climbing	1
Simulated Annealing	1
Custom/heuristic methods	16 out of 68
N/A	4 out of 68

* *The sum of occurrences is greater than 68, as some studies consider more than one algorithm.*

Custom/heuristic methods. In 16/68 studies, neither a general-purpose exact method nor a conventional metahueristic is adopted, but researchers proposed custom procedures for solving the TRAP tailored for the specific formulation. For instance, Monden et *al.* proposed several allocation strategies based on complexity metrics and predicted faults, and use an ad-hoc solution for them (Monden et al., 2013); the studies by Felderer et *al.* on risk-based testing adopt a risk matrix as scheme for the allocation algorithm (e.g., Felderer and Ramler, 2014a; Ramler and Felderer, 2015; Felderer and Ramler, 2014b); the works by Amrita et al. exploit fuzzy logic to allocate test cases based on operational profile (Amrita and Yadav, 2015b; 2015a).

The main findings about the type solution of TRAP models are reported in the following.

F9 *Most of TRAP studies is solved by exact methods. All the studies using exact methods focus on solving SO problems, while no MO study is solved by exact methods.*

F10 *Although SO studies are the vast majority, if we restrict the scope to works solved by metaheuristics, there is a balance, since MO problems are difficult to handle without search techniques: 10 SO and 8 MO studies.*

F11 *When researchers opt for exact methods, those based on Lagrangian multipliers are by far the most common way to solve the*

testing resource allocation NLPP. When search-based techniqeus are preferred, a wide variety of metaheuristics is tried — 12 different MO and 7 SO metaheuristics in 18 papers.

Differently from exact methods, studies adopting metaheuristics tend to compare different algorithms. We looked at how many times an algorithm performed better than another, when this information is available.

Comparisons are run more often in MO studies (6/8), while just 4/10 SO studies report a comparison. In the MO cases, metaheuristics are compared to each other by conventional coverage, convergence and diversity metrics (e.g., hypervolume, (inverse) generational distance, spread) and, in few cases, by also problem-specific metrics (i.e., which algorithm yields the best solution for a given objective function or the best cost-optimality trade-off). In the SO cases, a metaheuristic is compared either against another metaheuristic (e.g., GA vs Simulated Annealing) or against problem-specific algorithms (e.g., the *K-A, Y-X, T-M* algorithms — from initials of authors' name — for series-parallel archtiectures) by problem-specific metrics (e.g., related to the maximization of faults or reliability, or minimization of failure rate, cost, time or effort).

In the MO case, NSGA-II performed better than competing metaheuristics in 3 out of 7 cases. HaD loses a comparison in 4 out of 5 cases. Algorithms based on DE (MODE, WNS-MODE) wins a comparison in 2 out of 3 cases, while the hybrid CellDE looses 1 out of 1 times. PAES, IBEA, SPEA2, SMPSO, and MOCell lose their comparisons (7 cases in total). MOEA/D, RWGA and MODE are compared 1 time and win the comparison against the others.

As for the SO case, GA performed better than problem-specific algorithms, such as the Y-X and T-M algorithms (Gao and Xiong, 2015), in the considered cases, and GLSA outperformed Simulated Annealing, while Hill Climbing outperformed a greedy-based policy. Moreover, in the two cases in which it is used, the effect of local search was found to be beneficial (Gao and Xiong, 2015; Shuaishuai et al., 2013). The former reports the formulation of the discussed GLSA variant of GA in a single-objective setting; while the latter adopts a local search strategy to improve multi-objective metaheurstics: MOEA/D, HaD, NSGA-II

4.4. Evaluation/validation

Table 7 reports the results with respect to the *validation* dimension and its attributes.

Type of validation. Most of the times, researchers use a numerical analysis to validate the TRAP model (55/68). These are conducted either on illustrative datasets and toy examples (in 48 cases) or, more rarely (in 7 cases), on real data taken from the literature or past experiences. In few cases the proposed method is used in practice, with only 7 studies reporting an experiment with the involvement of industry (4 industrial case studies and 3 action research studies). In 6 studies, a real system is used, but in lab contexts (3 academic case studies, 2 lab experiments and 1 simulation used for empirical evaluation). Controlled experiments, as well as prototyping, are never used by researchers. From this picture, a strong need for evaluation research emerges as means to favour the applicability of TRAP models and solutions in practice.

Dataset. This attribute refers to how the dataset used for validation is derived. A real dataset is considered by 20/68 studies: these are either derived from the real system used as case study or taken from the literature from papers reporting historical failure data tracked on a real system. In all the other cases (48/68), and mostly in the numerical validation type, the dataset is generated artificially for the purpose of illustration.

Sensitivity analysis. A careful evaluation should consider how the proposed solution works in case the model's parameters vary and the model's assumptions are violated, especially with numerical illustrations. Sensitivity analysis is a useful tool for this purpose. However, a minority of studies exploit sensitivity analysis (45/68); rather, validation assumes a fixed set of parameters, and results refer to that specific setting only.

Problem size. Fig. 6 reports the (empirical) cumulative distribution of the number of modules considered in the validation (either from a real system and from a numerical example). The mean of the distribution is 9.5, the median is 8. It can be seen that most of the studies (39/68) consider a problem size between 5 and 15 modules. 16/68 studies consider less than 5 modules and the re-

Table 7
TRAP studies by *Evaluation/Validation*.

Evaluation/Validation	Frequency
Method	*n* out of 68
Mathematical/numerical analysis	55
Action research	4
Industrial case study	3
Academic case study	3
Lab experiments	2
Simulation as empirical method	1
Controlled experiments with practitioners	0
Prototyping	0
Real dataset	
No	48
Yes	20
Sensitivity analysis	
No	45
Yes	23
Problem size	Fig. 6
Industry involvement	
No	61
Yes	7
Tools	
No	60
Yes	8

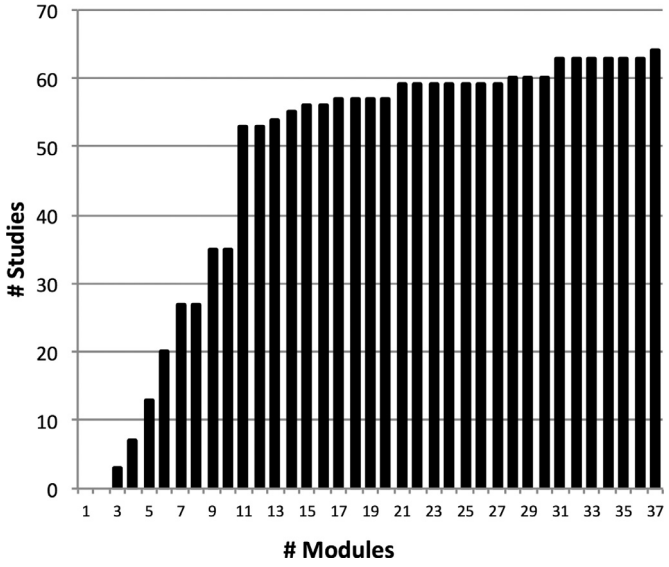

Fig. 6. Distribution of the TRAPs' size.

maining 13/68 consider more than 15 modules. The biggest problem size is 36 modules.

Industry involvement. In just 7/68 cases, the validation has been conducted in an industrial context, by either an industrial case study or an action research. Examples of industry include a case from Cisco Norway, where an allocation strategy is tested on a videoconferencing system (VCS) (Wang et al., 2016), and one at Selex ES Italy, a company developing mission-critical systems for Air Traffic Control (ATC) and Vessel Traffic Systems (VTS) (Carrozza et al., 2014).

Tools. In some cases (9/68), a tool is explicitly mentioned to solve the TRAP model. However, the mentioned tools are always based on well-known general-purpose tools for solving optimization problems, such as Lingo or Matlab.

A summary of findings and related comments follow.

F12 *The majority of studies use numerical illustrations and artificial dataset to validate their models.*

The relatively low number of real systems involved in the evaluation is partially due to the difficulties in experimenting a TRAP. First, the TRAP model should be built on historical testing data referring to each module (e.g., the inter-failure times, debugging cost data, etc.); then, the allocation solution computed by the TRAP model should be applied on the testing process of a successive release, with real testers on the real system; finally, results should be compared with the application of the same testing process under non-optimal allocation or at least checked against model's prediction. All this is expensive for industry and new ways for testing TRAP proposals in real scenarios are needed.

F13 *Although sensitivity analysis is a valuable tool to test the solution against parameters variation, it is not much used. Only 1/3 of the studies and less than half of the studies using numerical illustration conduct a sensitivity analysis.*

F14 *The problem size does not go beyond 36 modules in the considered studies.*

The latter finding represents a potential negative aspect regarding the evaluation of scalability. On one hand, it is reasonable to keep the granularity of modules rough: in fact, the TRAP model usually applies to modules that should be *independently testable* (if the test activities on a module were dependent on test activities of another module, that would affect the allocation solution — more complex models, capturing dependencies of test activities, would be needed). Therefore, assuming a small number of (independent) modules is not much unrealistic, as it can well represent real problems in many contexts. On the other hand, this does not elude the problem of scalability, because as systems grow in size and independence between modules is enforced (e.g., in service-oriented or microservice architectures), the size of TRAPs will rapidly grow. Scalability is likely to be an issue in the coming years.

Overall, the analysis shows that the TRAP research often opts for evaluation or validation approaches loosely connected with industry, with unrealistic scenarios and datasets, and no sensitivity analysis. From this point of view, a significant step forward is required to make the TRAP models practicable and useful.

5. Research directions

The above findings capture synthetically the current state of the art with reference to the analyzed dimensions. In the following, some research directions are outlined to suggest potential areas of investigation.

TRAP formulation. Most of current TRAP proposals are based on reliability, cost and effort/time models that do not accurately reflect the modern development, testing and debugging practices. For instance, development practices like *continuous integration* (CI) or *DevOps* demand for a new view of test planning and resource allocation, where conventional SRGMs need to be revised to account for a highly dynamic context with a different testing and release process, as the impact of assumption's violation is much more severe in this new scenarios. Indeed, in such contexts, the immediate feedback coming from runtime data can be a valuable support to build new SRGMs, closer to the system-in-operation, and enable a "continuous" tests allocation exploiting near real-time data for accurate modelling. New SRGMs to model reliability growth of a continuously evolving software are needed for these processes, or, even better, alternative ways to describe the testing-reliability relation, such as non-parametric models, which are known to make less (and to be less sensitive to) assumptions. Testing cost/effort models are also very different in a CI organization, and further research is needed to appropriately model them for TRA purposes.

In a similar way, architectural models in a TRAP should be aligned to current trends. The architectural style of software systems is by now migrated toward a dynamic approach, wherein the inter-relation among modules is no longer established statically at design time. Applications are thought to evolve, with loosely coupled modules that dynamically interact with each other at runtime (e.g., web services, microservices). This opens to the possible exploitation of dynamic inference techniques to extract architectural models at runtime and periodically re-allocate resources for testing.

Looking at the organizations, a detailed modelling of objectives and constraints is needed to have context-tailored models rather than one-fits-all solutions. Models are still over-simplistic with respect to real production processes, especially those of large organizations. For instance, debugging processes have more complex workflows than the ones assumed in a TRAP. In fact, the cost of debugging can be hardly captured by a constant parameter, because it usually depends on how each phase in the workflow is managed as well as on *human factors* like debuggers' skills, experience, attitude. Each activity of the workflow should be properly modeled, e.g., the bug queuing process, the bug-debugger assignment policy (e.g., select debugger based on bug severity and debugger experience), the bug resolution and closure.

Finally, traditional objectives are reliability, cost, time; however, several quality attributes are worth exploring, such as *power consumption* (of great importance for *cloud-native* applications), *security* or *performance*, which are key market drivers in today's software industry.

TRAP solution. The need for modelling dynamic contexts entails the need for developing dynamic solutions. A TRAP solution method should be able to control the uncertainty caused by the development and operational environment. A possibility is to adopt a dynamic allocation strategy, namely to solve the TRAP model more times as testing proceeds, like the works discussed in Section 4.2. More broadly, adaptive allocation strategy, for instance exploiting dynamic search-based metaheursitics, is a direction worth to be explored.

Another option is to *tolerate* uncertainty. While it is undebatable that uncertainty should be reduced (e.g., by more accurate models), a share of uncertainty is inevitable: *robust optimization* is a way of dealing with that, as it considers the uncertainty associated with models' parameters (e.g., in terms of confidence interval) and provides *ranges* of solutions (i.e., interval-solutions instead of point-solutions). An example can be found in our recent work (Pietrantuono et al., 2017).

The potential of metaheuristics for managing problems at a large scale can be exploited to accomodate some of the emerging needs described above (e.g., more complex models for testing/debugging process, many-objective optimization, the possible increase of problems size). A challenge in the mentioned contexts

(e.g., CI or microservices) is about the efficiency of metaheuristics, since a "continuous" optimal allocation would require a fast feedback. On the other hand, exact methods for affordable MO problems should be explored, and more studies are needed on exact methods, since in contexts where the error on allocated resources can have a sever impact (e.g., where the cost of testing and/or the cost of failures is high) an exact solution would be needed.

TRAP evaluation/validation. For TRAP models to increase their popularity and usefulness, more real-world experiences are needed, as resulting from Section 4.4. Applying models to concrete instances, especially in industry, allows testing assumptions, to figure out which ones are realistic and which call for new, more comprehensive, models. However, in many cases, it is not possible to "drive" the real testing process, but just to observe it, making it difficult to apply a solution suggested by a TRAP model in a next testing session. In such a context, causality analysis may help to identify most impactful factors in a process/organization and their relation with the testing objective to optimize. An alternative to reduce the cost is to run *a posterior* validation: results of the actual non-optimal allocation performed in the real testing process (e.g., in terms of detected faults) is compared with the *hypothetical* allocation proposed by the TRAP solution (what would have happened if the TRAP allocation were used, e.g., in terms of faults found in each module) (Carrozza et al., 2014). Results would be indeed more realistic than numerical validation, as the assumptions underlying a TRAP (e.g., assumptions of SRGMs) are tested in a real context, and the impact of their violation is assessed without necessarily doing a sensitivity analysis. A further direction is to develop *tools* or *simulators* able of representing, by proper models, a high variety of contexts in terms of testing objectives and constraints, software architecture, testing/debugging as well as development processes, organizational features.

Currently, the aspect of realistic evaluation/validation represents the main hurdle for the future development of TRAP research, as it causes a deep gap between theory and practice. New models can be conceived and new exact or approximate solutions can be devised: however, the TRAP is an engineering problem with a huge economical impact in software industry, because of its implications on the testing process, which is, and will likely be, a key driver for high-quality software development. The TRAP research has big opportunities to impact, but researchers will catch them inasmuch as they will connect with industry and its real-scale problems.

6. Guidelines

The following main aspects need to be considered for applying an existing (or developing a new) TRAP model and solution:

- *Testing objectives/constraints*. Indeed, the testing cost and time/effort are the two aspects that always matter, as either objective or constraint. Then, depending on the quality requirements, practitioners may want to improve reliability (either in terms of number of defects or *operational* reliability, which are different form each other (Cotroneo et al., 2016)), security, performance, energy consumption or machine cost when performing testing in the cloud (i.e., testing-as-a-service), or more than one together. The objective calls for the second important choice: modelling the relationship between the chosen objective(s) and the testing effort devoted to it. Some of the existing models (e.g, SRGM) can be considered, but testing-quality relation models for security, performance or other attributes are not common – in such cases, new models need to be constructed, by inferring how the quality attribute of interest changes when more testing is devoted to it. However, even opting for existing SRGMs, the model needs to be validated and possibly adapted for the specific context: for instance, in a continuous deployment scenario, where the software and the environment continuously evolves, existing models may not work well. One should prefer models relying less on those assumptions related to "stability" (e.g., the software, the environment or the usage of the software do not change over time). In this sense, SRGMs with multiple change points, or even non-parametric and/or Bayesian models, can better suite the current needs for continuously evolving contexts.
- *Architectural description*. The second choice is about how to describe the inter-components relationship. The minimum level of granularity needs to be chosen considering the smallest units that are testable independently, namely those to which a tester can allocate resources and run testing without interfering with testing of other units. The relation can be captured by architecture-based stochastic models (Goševa-Popstojanova and Trivedi, 2001), either path-based or state-based. Such Markovian models work better if the inter-components dependencies are minimized; when a tighter relation between components is present, more complex solutions should be adopted, such as composite and hierarchical models (Gokhale and Trivedi, 1999). Dynamically evolving architectures, like web services and microservices, require either exploiting monitoring data to continuously update the architectural data or to periodically re-allocate resources for testing.
- *Debugging and fault tolerance*. Additional aspects to consider in the modelling framework, which by now have become impactful, are: the debugging process (namely, the non-negligible debugging time and the bugs re-introduction rates – models exist for both aspects, but just for reliability) as well as the inclusion of fault tolerance (e.g., architectures like microservices foresees patterns for fault tolerance that should be included in the architectural modelling, as they can significantly impact the optimal testing allocation).
- *Information extraction* to parametrize the models. In today's software, design-time models are likely to become obsolete soon; hence, models need to be updated at runtime. Compared to the past, there are two sources of information that should be exploited: field data and data from the organization (e.g., issue trackers), which are both widely available in today's systems. Field data gathered from monitoring are paramount to update both the objective models and the architectural models. Issue trackers contain defect data (programming bugs, vulnerabilities, performance issues), useful to update the objective models (e.g., the SRGMs).
- *Solution*. Depending on the objectives, the size of the problem and the criticality of the testing allocation decision, one can opt for metaheuristics (many of them are available) especially for multi/many-objective optimization or by exact methods. The latter should be preferred when the error on allocated resources can have a sever impact (e.g., where the cost of testing and/or the cost of failures is high).
- *Evaluation/Validation*. Models should be validated in the organization where they are going to be used. The main reason is that testing involves a significant amount of human intervention and depends a lot on organizational factors. For instance, the testing-reliability relation may vary in relation to the experience and skills of testers, as well as the debugging time, the re-introduction bug rates, the testing effort function. The relative complexity of the system components may impact testing in different ways, as well as the development team that implemented one or another component (e.g., some may be outsourced). The organization can take decisions that impact the testing process and the test-

Table 8
Checklist of activities for SM self-assessment, from Petersen et al. (2015).

Phase	Action	Applied
Need for map	Motivate the need and relevance	√
	Define the objectives and RQs	√
	Consult target audience with RQs	•
Study identification	*Choosing search strategy*	
	Snowballing	√
	Database search	√
	Manual search	•
	Develop the search	
	PICO	•
	Consult libraries or experts	•
	Iteratively try finding more relevant papers	•
	Keywords from known papers	√
	Use standards, encyclopaedias and thesaurus	•
	Evaluate the search	
	Test-set of known papers	√
	Experts evaluate results	•
	Search web-pages of key authors	√
	Test-retest	•
	Inclusion and exclusion	
	Identify objective criteria for decision	√
	Add additional reviewer, resolve disagreements	•
	Decision rules	•
Data extraction and class.	*Extraction process*	
	Identify objective criteria for decision	√
	Obscuring information that could bias	•
	Add additional reviewer, resolve disagr.	•
	Test-retest	•
	Classification schemes	
	Research type	√
	Research method	√
	Venue type	√
Validity discussion	Validity discussion/limitations provided	√

ing teams' performance. Therefore, while simulation and numerical analysis are important for the initial tuning of the model, then an evaluation phase in-the-field is needed in order for models to provide a satisfactory predictive accuracy. If possible, the support of tools integrated with other facilities within industry (e.g., with the issue tracker, or with monitoring tools) could make everything more transparent to testers, who would just query the tool for the best allocation, given a budget to spent. Unfortunately, to date, little support is offered by the current state-of-the-practice, but existing modelling frameworks (e.g., for stochastic modelling) and tools for optimization problems solution (e.g., for metaheuristics and exact methods) can be borrowed for creating customized facilities.

- *Return of investment*. Before setting up an organizational structure to *i)* gather data, *ii)* to formulate, refine and apply the TRAP models, and finally *iii)* to plan the activities and organize a team according to the TRAP solution suggestions, one needs to assess if the benefit is worth the cost. The need for a quantitative testing resource allocation emerges in a more pronounced way when an organization deals with large (modular) software systems, in which the testing cost represents a large part of the production budget and in which multiple testing teams can work in parallel on several independently testable units. Moreover, the above-described models are by far more accurate and incisive if, like in large companies, the development/testing processes and personnel organization are well-structured and, more importantly, stable with time. Contrarily, the models do not make much sense when the system to be tested is small or managed by a single tester, and/or whenever the company organization or development/testing process is subject to sudden changes, in which case the *history* needed to create reliable models is limited or, at least, not representative. This makes the lack of evaluation/validation on large systems and in real industrial contexts, where the TRAP makes more sense, a crucial missing aspect, as just discussed above.

7. Threats to validity

In the following, the main threats to validity are reported. The process of selecting the primary studies was performed by a single person implying that some papers might have been included or excluded incorrectly. As pointed out by Wohlin et al. (Wohlin et al., 2013), two mapping studies of the same topic are likely to end up with different sets of articles. To mitigate this threat and have a representative sample, a well-defined search strategy has been followed, consisting of both automatic search and snowballing on the selected studies, followed by a search evaluation step in accordance with Petersen's guidelines (Petersen et al., 2015) and Kitchenham's suggestions (who also conducted a single-author SM) (Kitchenham, 2010). The test-set approach for search validation suggested that there might be papers that have been omitted, which led me to create two validation sets for double checking the search. Also, the search was conducted by querying multiple data sources, so as to cover a high number of publishers. Additionally, the terms used in the search string were very general. This allowed to be very conservative in the first search (which included 542, 981 and 157 from the three libraries) so as to not miss relevant studies. The resulting set was then manually filtered to discard irrelevant results up to 265 studies. Moreover, a set of documented inclusion and exclusion criteria was used to refine the search unambiguously.

Another threat can regard the quality of selected studies. To mitigate this threat: *i)* only peer-reviewed studies have been considered (excluding the so-called grey literature, e.g., white papers,

editorials, etc.); *ii)* studies are searched among those indexed by the most used digital libraries in computer engineering and computer science, which filter out several low-quality conferences and journals; *iii)* inclusion/exclusion criteria applied on each of the 265 studies after full text reading assured to keep only studies pertinent to TRAP; *iv)* a quality assessment step has been carried out to give an overview of the quality of selected studies.

Data extraction and classification could also be biased by a subjective interpretation. To reduce such a risk, we used again the test-set approach, to have a template and a classification scheme to categorize the studies unambiguously. The classification scheme has been derived by iteratively refining an initial scheme as more and more studies were analyzed, so as to guarantee that the data extraction process was aligned with the research questions. In general, the best practices for mapping studies by Petersen et al. have been followed in each phase of the study as documented above (Petersen et al., 2015).

A further threat is represented by the sample size, $N = 68$. In an SM, one is not supposed to read in detail all the papers compared to a systematic literature review (SLR), hence having a greater risk for misclassification but mitigated by a larger sample size. However, although the goal of this study is exactly that of an SM (i.e., to examine the research activity, summarize the main findings, and identify gaps, which would not require in-depth reading of the papers), the relatively few studies published in this area allowed me to go much deeper in each study, also carrying out quality assessment. This, along with the support of the detailed procedure explained above, had the side effect of reducing the risk of misclassification. Finally, to evaluate the whole SM process, I conducted a self-assessment. Table 8 reports the actions deemed relevant to conduct systematic mapping studies, proposed by Petersen in his guidelines (Petersen et al., 2015). Based on how many and what actions are conducted, a self-assessment is carried out to evaluate the mapping process. Petersen proposes a set of rubrics, one per phase/subphase (*need for the map, search strategy, search evaluation, data extraction and classification, validity discussion*), by which a score is assigned (0 to 1, 2 or 3, depending on the phase), according to the number of actions taken per (sub)phase, distinguishing the cases of: *no evaluation, minimal evaluation, partial evaluation* or *full evaluation* for each of the 5 phases. Considering the percentage of actions taken with respect to all the potential alternatives for each step as a quality score, this study scored 50%, which is above the score of many SMs in software engineering: the Petersen's study scored 31% and the median over the 52 mapping studies reviewed by Petersen is 33%. Following the evaluation rubrics in Petersen et al. (2015), the scores per phases are: *need for map: 1; search strategy: 1; search evaluation: 2* ; *extraction and classification: 2; validity: 1*, the total being 7 — the study by Petersen scored '1' in all the phases, thus 5.

8. Conclusion

This article presented an analysis of the existing literature on the TRAP research area. On a set of 68 studies, the work investigates the ways in which the allocation problem is formulated analytically, what models are preferred, what methods are developed for their solution, and what approaches researchers adopt for validation. The analysis allowed to spot areas in which further research is required, also considering the modern software along with its development processes. The main research directions to address the gap have been outlined with reference to the development of new and more suitable formulations as well as to the need of real-world (industrial) experiences to advance the state-of-the-art and state-of-the-practice together. Results are expected to benefit researchers and practitioners working in the testing area and in the software reliability area, especially those dealing with large component-based systems.

Declaration of Competing Interests

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.

References

Amrita, Yadav, D.K., 2015. A novel method for allocating software test cases. Procedia Comput. Sci. 57, 131–138. 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015).

Amrita, Yadav, D.K., 2015. Operational profile based software test case allocation. 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom) 1775–1779.

Arksey, H., O'Malley, L., 2005. Scoping studies: towards a methodological framework. Int. J. Soc. Res. Methodol. 8 (1), 19–32.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. 1 (1), 11–33.

Boehm, B., 1981. Software Engineering Economics, first Prentice Hall PTR.

CALP, M. H., KÖSE, U., 2018. Planning activities in software testing process: aliterature review and suggestions for future research.

Carrozza, G., Pietrantuono, R., Russo, S., 2014. Dynamic test planning: a study in an industrial context. Int. J. Software Tools Technol. Trans. 16 (5), 593–607.

Catal, C., Diri, B., 2009. A systematic review of software fault prediction studies. Expert Syst. Appl. 36 (4), 7346–7354.

Cinque, M., Cotroneo, D., Pecchia, A., Pietrantuono, R., Russo, S., 2017. Debugging–workflow-aware software reliability growth analysis. Softw. Test. Verif. Reliab. 27 (7).

Cinque, M., Gaiani, C., De Stradis, D., Pecchia, A., Pietrantuono, R., Russo, S., 2014. On the Impact of Debugging on Software Reliability Growth Analysis: A Case Study. In: Murgante, B. (Ed.), Computational Science and Its Applications–ICCSA 2014. Springer International Publishing, pp. 461–475.

Cotroneo, D., Pietrantuono, R., Russo, S., 2013. Combining operational and debug testing for improving reliability. IEEE Trans. Reliab. 62 (2), 408–423.

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. doi:10.1109/TSE.2015.2491931.

Elberzhager, F., Rosbach, A., Münch, J., Eschbach, R., 2012. Reducing test effort: a systematic mapping study on existing approaches. Inf Softw Technol 54 (10), 1092–1106.

Elegbede, A.O.C., Chu, C., Adjallah, K.H., Yalaoui, F., 2003. Reliability allocation through cost minimization. IEEE Trans. Reliab. 52, 106–111.

Felderer, M., Ramler, R., 2014. Integrating risk-based testing in industrial test processes. Softw. Qual. J. 22 (3), 543–575.

Felderer, M., Ramler, R., 2014. A multiple case study on risk-based testing in industry. Int. J. Softw. Tools Technol. Trans. 16 (5), 609–625.

Fiondella, L., Gokhale, S., 2012. Optimal allocation of testing effort considering software architecture. IEEE Trans. Reliab. 61 (2), 580–589.

Gao, R., Xiong, S., 2015. A genetic local search algorithm for optimal testing resource allocation in module software systems. In: et al., D.-S. H. (Ed.), Intelligent Computing Theories and Methodologies. Springer International Publishing, pp. 13–23.

Garousi, V., Mäntylä, M.V., 2016. A systematic literature review of literature reviews in software testing. Inf. Softw. Technol. 80, 195–216.

Goel, A.L., 1985. Software reliability models: assumptions, limitations and applicability.. IEEE Trans. Softw. Eng. SE-11 (12), 1411–1423.

Goel, A.L., Okumoto, K., 1979. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Trans. Reliab. R-28 (3), 206–211.

Gokhale, S., Trivedi, K., 1998. Log-logistic software reliability growth model. In: Proc. 3rd Int. High-Assurance Systems Engineering Symposium (HASE), pp. 34–41.

Gokhale, S.S., Trivedi, K.S., 1999. A time/structure based software reliability model. Ann. Softw. Eng. 8 (1–4), 85–121.

Goševa-Popstojanova, K., Trivedi, K.S., 2001. Architecture-based approach to reliability assessment of software systems. Perform. Eval. 45 (2), 179–204. Performance Validation of Software Systems.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2012. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. 38 (6), 1276–1304.

Huang, C.-Y., Kuo, S.-Y., Lyu, M., 2007. An assessment of testing-Effort dependent software reliability growth models. IEEE Trans. Reliab. 56 (2), 198–211.

Huang, C.-Y., Lo, J.-H., 2006. Optimal resource allocation for cost and reliability of modular software systems in the testing phase. J. Syst. Softw. 79 (5), 653–664.

Huang, C.-Y., Lo, J.-H., 2006. Optimal resource allocation for cost and reliability of modular software systems in the testing phase. J. Syst. Softw. 79 (5), 653–664. Quality Software.

Huang, C.-Y., Lo, J.-H., Kuo, S.-Y., Lyu, M.R., 2004. Optimal allocation of testing-resource considering cost, reliability, and testing-effort. In: 10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004. Proceedings., pp. 103–112.

Huang, C.-Y., Lyu, M., 2005. Optimal release time for software systems considering cost, testing-effort, and test efficiency. IEEE Trans. Reliab. 54 (4), 583–591.
Huang, C.-Y., Lyu, M.R., 2005. Optimal testing resource allocation, and sensitivity analysis in software development. IEEE Trans. Reliab. 54 (4), 592–603.
Khan, M.G.M., Ahmad, N., Rafi, L.S., 2016. Determining the optimal allocation of testing resource for modular software system using dynamic programming. Commun. Stat. - Theory Methods 45 (3), 670–694.
Kitchenham, B., 2010. What'S up with software metrics?–a preliminary mapping study. J. Syst. Softw. 83 (1), 37–51. SI: Top Scholars.
Kitchenham, B., Brereton, P., 2013. A systematic review of systematic review process research in software engineering. Inf. Softw. Technol. 55 (12), 2049–2075.
Kitchenham, B.A., Budgen, D., Brereton, O.P., 2010. The value of mapping studies: a participantobserver case study. In: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering. BCS Learning & Development Ltd., Swindon, UK, pp. 25–33.
Kuo, W., Prasad, V.R., Tillman, F.A., Hwang, C.-L., 2001. Optimal Reliability Design: Fundamentals and Applications, 1st Cambridge University Press.
Kuo, W., Wan, R., 2007. Recent advances in optimal reliability allocation. IEEE Trans. Syst. Man Cybern. - Part A 37 (2), 143–156.
Lee, J., Kang, S., Lee, D., 2012. Survey on software testing practices. IET Software 6 (3), 275–282.
Lo, J.-H., Huang, C.-Y., 2006. An integration of fault detection and correction processes in software reliability analysis. J. Syst. Softw. 79 (9), 1312–1323.
Lo, J.-H., Kuo, S.-Y., Lyu, M.R., Huang, C.-Y., 2002. Optimal resource allocation and reliability analysis for component-based software applications. In: Proceedings 26th Annual International Computer Software and Applications, pp. 7–12.
M. Lyu (Ed.), Handbook of Software Reliability Engineering, McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.
Lyu, M.R., Rangarajan, S., van Moorsel, A.P.A., 1997. Optimization of reliability allocation and testing schedule for software systems. In: Proceedings The Eighth International Symposium on Software Reliability Engineering, pp. 336–347.
Lyu, M.R., Rangarajan, S., van Moorsel, A.P.A., 2002. Optimal allocation of test resources for software reliability growth modeling in software development. IEEE Trans. Reliab. 51 (2), 183–192.
Malhotra, R., 2015. A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. 27, 504–518.
Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M., Matsumoto, K., 2013. Assessing the cost effectiveness of fault prediction in acceptance testing. IEEE Trans. Softw. Eng. 39 (10), 1345–1357.
Musa, J.D., Okumoto, K., 1984. A logarithmic poisson execution time model for software reliability measurement. In: Proceedings of the 7th International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 230–238.
Nakagawa, Y., Miyazaki, S., 1981. Surrogate constraints algorithm for reliability optimization problems with two constraints. IEEE Trans. Reliab. R-30 (2), 175–180.
Ohishi, K., Okamura, H., Dohi, T., 2009. Gompertz software reliability model: estimation algorithm and empirical validation. J. Syst. Softw. 82 (3), 535–543.
Ohtera, H., Yamada, S., 1990. Optimal allocation and control problems for software-testing resources. IEEE Trans. Reliab. 39 (2), 171–176. doi:10.1109/24.55878.
Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. 64, 1–18.
Pietrantuono, R., Potena, P., Pecchia, A., Rodriguez, D., Russo, S., Fernandez, L., 2017. Multi-objective testing resource allocation under uncertainty. IEEE Trans. Evol. Comput. PP (99).
Pietrantuono, R., Russo, S., 2018. Search-based optimization for the testing resource allocation problem: research trends and opportunities. In: Proceedings of the 11th International Workshop on Search-Based Software Testing. ACM, New York, NY, USA, pp. 6–12.
Pietrantuono, R., Russo, S., Trivedi, K., 2010. Software reliability and testing time allocation: an architecture-based approach. IEEE Trans. Softw. Eng. 36 (3), 323–337.
Ramler, R., Felderer, M., 2015. A process for risk-based test strategy development and its industrial evaluation. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 355–371.
Rice, W.F., Cassady, C.R., Wise, T.R., 1999. Simplifying the solution of redundancy allocation problems. In: Annual Reliability and Maintainability. Symposium. 1999 Proceedings (Cat. No.99CH36283), pp. 190–194.
Shuaishuai, Y., Dong, F., Li, B., 2013. Optimal testing resource allocation for modular software systems based-on multi-objective evolutionary algorithms with effective local search strategy. In: IEEE Workshop on Memetic Computing (MC). IEEE, pp. 1–8.
Tillman, F.A., Hwang, C., Kuo, W., 1977. Determining component reliability and redundancy for optimum system reliability. IEEE Trans. Reliab. R-26 (3), 162–165.
Wang, S., Ali, S., Yue, T., Bakkeli, Ø., Liaaen, M., 2016. Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 182–191.
Wieringa, R., Maiden, N., Mead, N., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11 (1), 102–107.
Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., Engstrom, E., do Carmo Machado, I., de Almeida, E.S., 2013. On the reliability of mapping studies in software engineering. J. Syst. Softw. 86 (10), 2594–2610.
Yamada, S., Hishitani, J., Osaki, S., 1993. Software-Reliability growth with a weibull test-Effort: A Model & application. IEEE Trans. Reliab. 42 (1), 100–106.
Yamada, S., Ichimori, T., Nishiwaki, M., 1995. Optimal allocation policies for testing-resource based on a software reliability growth model. Math. Comput. Model. 22 (10), 295–301.
Yamada, S., Ohba, M., Osaki, S., 1983. S-Shaped reliability growth modeling for software error detection. IEEE Trans. Reliab. R-32 (5), 475–484.
Yang, B., Hu, Y., Huang, C.-Y., 2015. An architecture-based multi-objective optimization approach to testing resource allocation. IEEE Trans. Reliab. 64 (1), 497–515.
Yang, X., Tang, K., Yao, X., 2015. A learning-to-rank approach to software defect prediction. IEEE Trans. Reliab. 64 (1), 234–246.
Zachariah, B., Rattihalli, R.N., 2007. Failure size proportional models and an analysis of failure detection abilities of software testing strategies. IEEE Trans. Reliab. 56 (2), 246–253. doi:10.1109/TR.2007.895310.
Zhang, G., Su, Z., Li, M., Yue, F., Jiang, J., Yao, X., 2017. Constraint handling in nsga-ii for solving optimal testing resource allocation problems. IEEE Trans. Reliab. 66 (4), 1193–1212.

Roberto Pietrantuono is Assistant Professor at Federico II University of Naples, where he got his Ph.D. degree in Computer and Automation Engineering in 2009. His research interests are in the area of software reliability engineering, particularly in the software verification of critical systems, software testing, and software reliability analysis. He published more than 70 articles in relevant international journals and conferences. He is Senior Member of the IEEE.