Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Testing anticipatory systems: A systematic mapping study on the state of the art[☆]

Bernhard Peischl[b], Oliver A. Tazl[b,*], Franz Wotawa[a,b]

[a] Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber–Physical Systems (QAMCAS), Austria
[b] Institute for Software Technology, Graz University of Technology, Austria

A R T I C L E   I N F O

Article history:
Received 6 October 2020
Received in revised form 19 May 2022
Accepted 27 May 2022
Available online 4 June 2022

Keywords:
Anticipatory systems
Artificial intelligence
Software testing
Mapping study
Verification
Validation

A B S T R A C T

Context: Systems exhibiting anticipatory behavior are controlling devices that are influencing decisions critical to business with increasing frequency, but testing such systems has received little attention from the artificial intelligence or software engineering communities.
Goal: In this article, we describe research activities being carried out to test anticipatory systems and explore how this research contributes to the body of knowledge. In addition, we review the types of addressed anticipatory applications and point out open issues and trends.
Method: This systematic mapping study was conducted to classify and analyze the literature on testing anticipatory systems, enabling us to highlight the most relevant topics and potential gaps in this field.
Results: We identified 206 studies that contribute to the testing of systems that exhibit anticipatory behavior. The papers address testing at stages such as context sensing, inferring higher-level concepts from the sensed data, predicting the future context, and intelligent decision-making. We also identified agent testing as a trend, among others.
Conclusion: The existing literature on testing anticipatory systems has originated from various research communities, such as those on autonomous agents and quality engineering. Although researchers have recently exhibited increasing interest in testing anticipatory systems, theoretical knowledge about testing such systems is lacking.

1. Introduction

In his famous essay "*Why Software is Eating The World*" (Andreessen, 2011), Marc Andreessen indicated that we are experiencing a dramatic and broad technological and economic shift in which he expects many industries to be disrupted by software. According to this essay, all technology that is required to transform industries through software is finally in place and can be delivered on a global scale. With software steering the car while entertaining its passengers, smart agents choosing how to distribute energy across the power grid, and software-powered drones delivering goods, we are not only surrounded by the internet of things (IoT), but are increasingly surrounded by systems based on artificial intelligence (AI) which rely on anticipatory behavior. Anticipation is a function in a component or device that enables decisions to be made based on predictions, expectations, or beliefs about the future. An anticipatory system is a system where the current state depends not only on a previous state, but also on future needs. It is a system that adapts itself, that learns, like living organisms (Rosen et al., 2012; Nadin, 2010). As software exhibiting such anticipatory behavior (Nadin, 2014, 2010; Kinsner, 2009) controls devices that may harm humans or influence decisions critical to business, the validation and testing of anticipatory systems is becoming an important aspect in the field of software and systems engineering (Wotawa et al., 2018; Mattos et al., 2017a). For example, automated driver assistance systems (ADAS) exhibit anticipatory functions, but these need to comply with safety requirements at the same time. Such systems are attracting an increasing amount of attention, not only from researchers but also from industry and standardization bodies, which are starting to address related engineering issues. For instance, the automotive industry has developed the SOTIF standard (Safety Of The Intended Functionality, ISO/WD PAS 12448), which outlines how to demonstrate the safe behavior of such systems.

Verification and validation (Beizer, 1990) are two of the most important activities performed during system development to assure system quality. Verification processes are carried out to assure that a system fulfills certain specifications, while validation processes are conducted to assure that the system has the functionality users are expecting. Hence, verification answers the question "Are we building the system correctly?", whereas

☆ Editor: Burak Turhan.
* Corresponding author.
E-mail addresses: bernhard.peischl@ist.tugraz.at (B. Peischl), oliver.tazl@ist.tugraz.at (O.A. Tazl), wotawa@ist.tugraz.at (F. Wotawa).

validation answers the question "Are we building the correct system?". Particularly when testing anticipatory applications, it is difficult to strictly distinguish between these two tasks, as such systems dynamically adapt to the environment after deployment (Baresi and Ghezzi, 2010). These systems anticipate future events and perform intelligent actions, and this adaption process might take place at runtime. Thus, the system reacts to uncertainties (e.g., unknown requirements) after the system development phase (Pejovic and Musolesi, 2015). This observation makes testing during the development time an extremely difficult, if not even impossible, task. We either have to restrict adaptation so that a set of good behaviors is not exceeded or it does not violate safety properties or fail to fulfill other important requirements. In the former case, verification methods may be still applicable.

The pressing need to test such systems is further supported when considering a recent example. In Tom Simonite's Wired article *Even Artificial Neural Networks Can Have Exploitable 'Backdoors'*,[1] the author describes a case where a neural network can be trained to behave differently when identifying a traffic sign with or without an attached post-it note. Such behavior can have severe consequences in real life, underlining the fact that exhaustive testing of anticipatory systems is critically important.

Because of the growing importance of anticipatory systems in practice, we are interested in testing such systems. Therefore, we collected information on state-of-the-art testing being conducted with anticipatory systems to clearly describe the current knowledge base and the research challenges that will need to be tackled in the future. This paper makes the following contributions:

Contribution

We reviewed the peer-reviewed literature on testing in anticipatory systems and examined the publication sources. This supports future research by enabling the results to be easily be accessed by other researchers. First, we investigated the state of the art in this emerging field. In particular, we analyzed the currently available contributions and characterized them with respect to the type of research, form of contribution, and systems investigated. Second, we describe the identified research trends qualitatively.

Background

Anticipatory systems are being developed and operated with increasing frequency in areas like ADAS, web-agents, chatbots. as well as smart factories. These systems exhibit specific capabilities, such as sensing, actuation, self-awareness, control, and autonomy. The emergence of data collection mechanisms has ensured that these systems can both provide continuous feedback in response to the actual behavior and learn from their interactions with their environment. This ability to learn over time enables such systems to anticipate desirable behavior and, thus, to react to environmental changes. Although promising application fields for anticipatory systems exist, such as automated driving functions, chatbots, and automating energy distribution, methodology has not yet been established to test such systems systematically. In this study, we placed a main focus on testing performed with anticipatory systems in the context of autonomous driving and ADAS. In this domain neural networks are often used, e.g., in perception system for object detection. We decided to reflect this in our literature review and added neural networks to the search string used in this study. Note that with this decision, we do not exclude other machine learning or testing approaches if other inclusion criteria are met. However, the study results may not comprise all papers dealing with machine learning or testing approaches not considered in the search string.

Organization

In Section 2, we outline our research design and present the three research questions that we addressed in this study. The query construction, analysis, and classification procedures that we performed are also discussed. In Section 3, we describe the filtering process that we conducted to identify relevant publications. The results of this mapping study are presented in Section 4. We then provide a systematic categorization of research, contribution, and system types and discuss the solution proposal we identified. In Section 5, we propose the implications of the findings, and we discuss related research, in Section 6. Threats to validity regardingidentified in this study are outlined in Section 7, and conclusions are presented in Section 8.

2. Research design

In this section, we describe the research design underlying our study. We followed the best practices in conducting mapping studies in software engineering, as outlined in Kitchenham (2007), Budgen et al. (2008) and Petersen et al. (2015). Furthermore, we incorporated the lessons learned from our preliminary study (Peischl et al., 2017). In the following sections, we briefly outline the procedure we applied and introduce our three major research questions. Then, we list the different services and instruments used to collect the relevant data. Finally, we describe the validity procedures applied.

Research method

Here, we outline the overall research approach taken as part of the implemented procedures. While carrying out this study, we designed a comprehensive research method that allowed us to obtain updates continuously. One of our goals was to develop an instrument that would enable us to identify the prevailing trends in this emerging research field. By applying the outlined research design, the study methods presented herein can be updated and refined by other researchers and used to conduct future studies.

Preliminary study

In summer 2017, we conducted a preliminary study (Peischl et al., 2017) to explore the state of the art in the field of testing anticipatory applications. In this study, we collected data from two sources. First, we conducted a query using the Scopus[2] document search function. Second, we considered articles that had been published in a workshop series, which specifically addressed how to create artificial intelligence synergies in software engineering (Harrison et al., 2012, 2013; Turhan et al., 2014, 2015; Minku et al., 2016). We retrieved about 300 articles and defined criteria which we then applied to clean this data set. We then carried out a selection process to obtain 53 relevant publications.

Extended study

This study has been inspired by other secondary studies conducted in domains such as software process improvement,

[1] https://www.wired.com/story/machine-learning-backdoors, last accessed 2nd Nov. 2017.

[2] See https://www.scopus.com, last accessed: 15th Mar. 2020.

Table 1
Search strings used to retrieve relevant publications.

Id	Search string	Subject of interest
S_1	AI or artificial intelligence	AI-enabled systems
S_2	SELF- or self-	self-* systems, self-organizing systems
S_3	Adaptive	Adaptive systems
S_4	Robustness or resilient or antifragile or anti-fragile	Robust, resilient, or anti-fragile systems
S_5	Cyber–physical	Cyber–physical systems
S_6	Anticipatory	Anticipatory behavior
S_7	Perception	Perception of environment and context
S_8	Neural network	Systems relying on neural-networks
C_1	(test or testing) and software	Publication focus on software testing
C_2	Functional or regression or acceptance	Testing focus on functional, regression or acceptance testing

Table 2
Inclusion and exclusion criteria applied in our study.

Criteria	Description
IC_1	Title, keywords, abstract refer to AI and testing
IC_2	Article presents topics related to software test and AI
EC_1	Article is not available in the English language
EC_2	Article is not in the field of computer science, math, or engineering.
EC_3	Full text of article is not available
EC_4	Authors, title, or abstract is not available
EC_5	Article occurred multiple times
EC_6	Articles dealt with built-in self test (BIST) or software-based self test (SBST) of micro controllers
EC_7	Articles primarily dedicated to self-testing of system on a chip (SoC)
EC_8	Articles not related to the field of computer science, e.g., articles from domains such as medicine, clinical engineering, computational biology, manufacturing, nuclear science, pharmacy, and education; such articles fulfill criteria EC_2 as these are classified as engineering sciences
EC_9	Articles dealt with AI techniques used to build software engineering/testing tools rather than engineering/testing of anticipatory applications
EC_{10}	Articles do not investigate anticipatory systems

automotive software, and the continuous deployment of software-intensive products (Kuhrmann et al., 2016; Haghighatkhah et al., 2017; Rodríguez P. Haghighatkhah, A. and Lwakatare, L.E. and Teppola, S. and Suomalainen, T. and Eskeli, J. and Karvonen, T. and Kuvaja, P. and Verner, J.M. and Oivo, M.). After conducting the initial study and analyzing the data obtained, we incorporated the lessons learned and collected further publications. The extended study differs from our preliminary work in several major aspects, namely, we used an extended set of search strings, introduced context selectors, introduced system types, and applied forward and backward snowballing techniques to complement the results of the initial literature review. This research enabled us to provide a detailed analysis of current trends in testing AI applications. Below, we describe the research questions, the collection procedures, the query construction and data, as well as the harmonization and selection procedures we applied to obtain the presented results.

The lessons learned from our initial survey particularly led us to use an extended set of keywords and apply additional inclusion and exclusion criteria. In Peischl et al. (2017), we restricted our search to literature published in the field of computer science and applied the search string S_1 (see Table 1). However, as our initial study indicated that numerous communities contribute to the body of knowledge, we extended the scope of our literature review to encompass the fields of computer science, engineering, and math. Due to the wider scope, we also complemented the set of queries by introducing additional strings (see Table 1 S_2 to S_7) and context selectors C_1 and C_2, as this allowed for more flexibility in the query construction process. Furthermore, we refined and complemented the exclusion criteria (see Table 2 EC_1 to EC_{10}) to increase the precision and relevance of our mapping study. In the initial study, we applied IC_1, IC_2, and EC_1 to EC_5. In this extended study, we decided to broaden the scope of the mapping study, as our research questions RQ1, RQ2, and RQ3 were formulated as general exploratory questions.

Research questions

Our objective is to provide a snapshot of the published research work on the subject of testing anticipatory systems. In this context, we considered testing as covering both verification and validation. Accordingly, we defined three research questions RQ 1, RQ 2, and RQ 3:

- RQ 1: What research and contribution types are found on the subject of testing anticipatory systems, and how many papers have been published for each type? This research question was developed to examine the pool of publications on and about testing anticipatory applications. In particular, we were interested in exploring the various types of research and contribution in this evolving field of research.

- RQ 2: What types of anticipatory systems have been addressed? This question was developed to explore the structure of the publication pool by characterizing these systems in terms of four key stages: context sensing, inferring context, predicting context, and intelligent actioning. We also related these system types to the research and contribution types described in the previous RQ.
- RQ 3: What are the trends and open issues regarding the testing techniques/methods for anticipatory systems? The last question was developed to investigate this focal point and enable us to identify gaps in order to design research on testing anticipatory applications in the future.

Literature review

In order to retrieve a representative set of publications, we identified a seed set with a literature database. Scopus claims itself to be the largest database of abstracts and citations,[3] so we performed an automated search and executed the following queries in this database.

Query construction

All three authors defined specific keywords of interest. These keywords were selected from the common terminology in testing and AI. The keywords of interest were identified during discussions with members of various research groups and in the preliminary study (see Section 2, Preliminary study). We structured the queries into general search strings $S_1 - S_8$ (see Table 1) and narrowed this result set by refining the search with context selectors C_1 and C_2 (see Table 1). We defined the search strings $S_1 - S_8$ according to the topic of interest, e.g., we were particularly interested in identifying publications that addressed the topics of self configuration, healing, learning, awareness, self organization, or self adaptation. Specifically, we identified publications reporting on systems exhibiting self-* properties (Babaoglu et al., 2005, S_2) or those on systems with specific extra-functional properties, such as adaptability (S_3), robustness, resilience, or anti-fragility (S_4). Furthermore, we included the term cyber–physical (S_5), as such system might exhibit anticipatory behavior (which we explicitly captured with string S_6). Finally, specific search strings were used to identify publications that described how the system perceived its environment, the system context (S_7), or the learning behavior with neural networks (S_8). Because we wanted to focus mainly on autonomous driving and ADAS, we decided to include the term neural network, which tends to be commonly used in this application area. We did not exclude other machine learning approaches in our study but did not mention them in the search string. Hence, the results may not comprise all papers dealing with machine learning in the context of anticipatory systems.

To refine the data set in favor of software testing, we employed the context selectors C_1 and C_2 to limit the review to publications related to functional software testing. A specific focus was placed on later stages in the life cycle, e.g., regression or acceptance tests. We chose not to include system tests among the context selectors, because we were interested specifically in testing only the anticipatory part of a system. However, we did not exclude papers using other types of testing like system testing from our study if they were part of the set of results obtained using our search string. If, for example, a publication contained information about functional and system testing, we included it because of its information about functional testing.

Selection criteria

To further improve the precision and relevance of the mapping study, we applied the inclusion (IC) and exclusion criteria (EC) listed in Table 2. It was necessary to fulfill that each inclusion criterion, i.e., that both IC_1 and IC_2 apply, and we removed the article under investigation from our set if at least one exclusion criteria could be met (EC_1 to EC_{10}[4,5]).

Snowballing

The literature review resulted in a set of publications (i.e., the result set) that we considered as a seed set for the subsequent backward and forward snowballing process (Mourão et al., 2017; Wohlin, 2014). To carry out this process, we basically applied the process suggested by Wohlin (2014).

Backward snowballing

Backward snowballing refers to using the reference list to identify new papers to include in the results set. For this purpose, we reviewed the reference list in each article in the seed set and again applied the exclusion (EC_1 to EC_{10}) and inclusion criteria (IC_1, IC_2). We removed articles from the reference list that did not fulfill the inclusion criteria and those where at least one of the exclusion criteria was fulfilled. Next, we removed articles from the reference list that had already been examined, i.e., these had been identified earlier through either backward or forward snowballing in the ongoing or a previous iteration or were present in the seed set. If a paper was retained as a candidate for inclusion after applying the inclusion and exclusion criteria, we then examined where and how the article was referenced. The place and the context of the reference often provides information about the content of the candidate paper. In many cases, it proved more efficient to obtain this content information from the paper being examined than from the candidate paper directly.

Forward snowballing

Forward snowballing refers to identifying new articles based on papers that cite the article being examined. We used Google Scholar[6] to identify the citations for the paper under examination. For this purpose, we refine the search in Google Scholar to remove quotes and solely used citations. We examined each candidate article that cited the article under consideration. First, we examined the paper title and abstract and then applied the inclusion and exclusion criteria. If this proved insufficient, we examined the place citing the paper that had already been included. If it was still impossible to make a decision regarding inclusion or exclusion, we examined the full text of the identified paper.

After conducting backward and forward snowballing, we compiled a selection of the newly identified papers. These were then used for the next iteration. We continued to execute this backward as well as forward snowballing process until no new papers were found.

Attribute extraction and synthesis

Our study was carried out to provide a foundation for future study updates (e.g., to confirm the prevailing trends and to identify new trends). In order to increase transparency, ensure reproducibility, and broaden the scope of the study, we developed a procedure for collecting the publications for classification and analysis, which is described below.

[3] See https://www.elsevier.com/en-gb/solutions/scopus, last accessed: 1. Aug 2021.

[4] Unlike in the literature review, EC_2 was applied manually.

[5] Excluding EC_5 refers to multiple occurrences, i.e., the article was retained in the result set.

[6] www.scholar.com.

Table 3
Research types applied in our study (Wieringa et al., 2006).

Research type	Description
Evaluation research	An implementation has been carried out, evaluation of implementation has been conducted
Solution proposal	A solution of the problem is proposed, benefits/applicability are/is demonstrated by example, this includes proposals complemented by a demonstrating case study, however, no dissemination plan is obvious
Philosophical paper	Paper comes up with a new way of thinking or structuring a specific field, e.g., in the form of a taxonomy of conceptual framework, secondary studies like systematic literature reviews, or systematic mapping studies
Opinion paper	Captures a personal opinion, the work, however, is not grounded in related work or research methodology
Experience paper	Captures personal experiences and describes how things are done in practice

Table 4
Contribution types applied in our study (Kuhrmann et al., 2016).

Contribution type	Description
Model	Representation of observed reality by concepts, representation after conceptualization
Theory	Construction of a cause–effect relationship
Framework /method	Framework or method related to testing of anticipatory systems
Guideline	List of advice
Lessons learned	Number of outcomes from obtained results
Tool	A tool supporting testing of anticipatory systems

We first integrated and cleaned the set of results. This involved removing duplicates and harmonizing the set of results. Harmonization was necessary, as performing the forward and backward snowballing process in Google Scholar yielded citations that stemmed from various sources (e.g., ACM, IEEE Explore, Elsevier, and Scopus databases). We performed an analysis and classification procedure on this final data set. Thereby, we followed a voting process, whereby two researchers performed a classification of the papers according to their research, contribution, and system types. If both researchers agreed upon the type assignment, the article was directly classified according to this scheme. Articles that caused disagreements were discussed in a workshop led by the authors.

Analysis and classification

We used the following analysis scheme to answer our research questions.

Research types

To classify the research type, we relied on the research type scheme proposed in Wieringa et al. (2006). Table 3 lists the research types applied to classify the articles in the publication pool.

Contribution types

In order to categorize the testing of anticipatory applications as described in the various publications, we followed the recommendations of Kuhrmann et al. (2016) and applied the contribution types listed in Table 4 to the final set of publications.

System types

In order to provide a more detailed perspective, we collected information about the system subject to the test or validation. The set of results contains papers from various application domains (e.g., automated and autonomous driving, robotics, expert-systems, medical devices, military applications, and space applications). To gain insights into the types of systems addressed in the articles, we developed a set of system types. Inspired by the work presented in Pejovic and Musolesi (2015), we identified four key stages of anticipatory systems: context sensing, inferring context, predicting context, and intelligent actioning. The authors of Butz et al. (2003) characterize such systems as "a process or behavior that depends on both past and present but also on predictions, expectations, or beliefs about the future". This behavior is considered as natural in the sense that it is deeply integrated with intelligence. Table 5 outlines the four stages of anticipatory systems and provides descriptions of these. The decision to apply these stages was motivated by the fact that screening the final result set revealed that a variety of systems were covered. Clustering these systems by topic would have resulted in the creation of too many clusters, which in turn had the potential to limit the analysis process. Furthermore, applying stages allowed us to assign multiple attributes to a system and reduced the risk of inaccurate classification.

In the following section, we discuss these key stages of anticipatory systems:

Context sensing. Various devices will only be integrated into our everyday lives, if these devices are able to perceive the environment (i.e., perceive context). The perception of context in this sense refers to the perception of situations and entities that

Table 5
Characterization of the system as proposed by Pejovic and Musolesi (2015).

System characterization	Description
Context sensing	System collects and processes sensor data
Inferring context	System extracts features from raw data, connecting features with higher level concepts
Predicting context	System builds models of future events, predicts behavior
Intelligent actioning	System constructs a decision framework derives actions from past, present, and future events to close the feedback loop

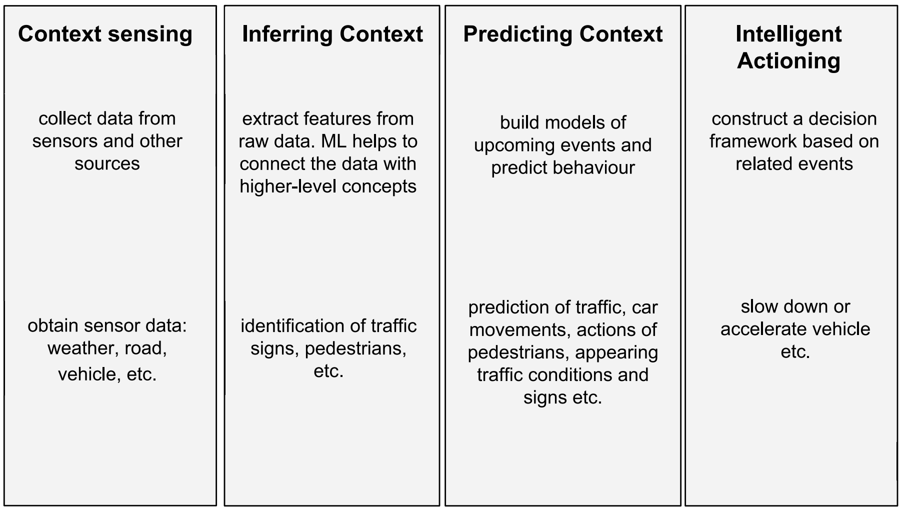

Context sensing
collect data from sensors and other sources
obtain sensor data: weather, road, vehicle, etc.
Inferring Context
extract features from raw data. ML helps to connect the data with higher-level concepts
identification of traffic signs, pedestrians, etc.
Predicting Context
build models of upcoming events and predict behaviour
prediction of traffic, car movements, actions of pedestrians, appearing traffic conditions and signs etc.
Intelligent Actioning
construct a decision framework based on related events
slow down or accelerate vehicle etc.

Fig. 1. Four key stages of anticipatory systems exemplified with a pedestrian detection system.

characterize the environment of the device. Such context may geographical, physical, social, temporal, or organizational aspects (see Pejovic and Musolesi (2015)). The sensed data serves as a basis for the next stages and determines the predictive capabilities with respect to the phenomenon of interest.

Inferring context. In order to infer context, a single sensor modality is rarely sufficient. Thus, this key stage typically relies on the context-sensing stage discussed previously. Inferring context involves identifying the most informative modalities and features of raw sensor data. For example, machine learning techniques can then be used to build a model of the phenomenon of interest. This includes addressing the challenge to select the appropriate representation for the sensor data and to select appropriate machine learning techniques (see Pejovic and Musolesi (2015)).

Predicting context. Today, the prediction of user behavior (e.g., mobility prediction, prediction of traffic conditions) is supported by a large set of multimodal data. For example, Eagle and Pentland (2006) accumulated traces from one hundred subjects monitored over a period of nine months. The New York yellow cab data set[7] contains trip records for trips made in Manhattan, Brooklyn, and Queens in 2013. Such data sets can be used to create dedicated models for prediction. In this key stage, we are building models of future events and predicting the behavior of users or things. This includes incorporating data from multiple users and multiple views.

Intelligent actioning. Intelligent actioning is the last key stage of anticipatory systems. In this stage, the system references past, current, and possible future events to make intelligent decisions. One of the core issues being faced in the development of anticipatory systems is how to enable systems to make decisions about these actions and how to close the feedback loop to the observable effects of these actions.

Example. A pedestrian detection system (see Fig. 1) is a typical example of an anticipatory system. This system needs to collect data from various sensors, e.g., obtain raw sensor data on the current weather, road conditions, and the state of the vehicle (key stage: context sensing). Some inference procedure needs to be performed to aggregate the raw data into higher-level concepts, which enable the system to identify traffic signs or pedestrians (key stage: inferring context). Furthermore, such a system will apply reasoning to predict possible future behavior, i.e., it will create a model of probable upcoming events that allows the system to anticipate the car movements and future traffic conditions (key stage: predicting context). Eventually, the software needs to take actions and, for example, raise an alarm and slow down or accelerate the vehicle (key stage: intelligent actioning).

From our initial study (Peischl et al., 2017), we learned that the literature on testing anticipatory systems is published by researchers in various scientific communities. As different concepts, abstractions, and wording might be used among the various scientific communities to describe the behavior of the system under test (SuT), we decided to introduce guidelines that could be applied to classify the system type. These guidelines have been successfully applied to ensure the high quality of this classification process. Table 6 outlines some of the guidelines we used to determine whether or not a paper contributes to a key stage and to identify this key stage.

3. Filtering and classification process

In this section, we discuss the filtering and classification processes used in our study.

[7] http://www.andresmh.com/nyctaxitrips, last accessed: 4th Nov. 2017.

Table 6
Quality instrument for determining the system type.

Guideline	Key stage	Description
G1	Context sensing	Does the SuT collect data from the environment? For example, this question can be answered by arguing that the SuT collects data from a specific geographical, physical, social, temporal, or organizational context.
G2	Inferring context	Does the SuT involve multiple sensors? Is a feature extraction or any other kind of abstraction carried out? For example, machine learning techniques are often used to associate raw data from various sensors with higher level abstractions.
G3	Predicting context	Does the SuT predict future behavior or events? For example, rules, plans, or other models are often used to anticipate future behavior.
G4	Intelligent actioning	Does the SuT implement some decision framework and carry out actions? For example, moving actuators might change the context. Does an action influence the previous stages, so that the SuT constitutes a feedback loop?

Table 7
Data collection and filtering of results. Queries retrieved on 10th Feb. 2020.

Search criteria	Automatic search		Manual selection	
	Hits	$EC_{1/2}$	$EC_{3/4}$	Disagreement
S_1 and C_1 and C_2	492	423	177	59
S_2 and C_1 and C_2	1021	636	166	57
S_3 and C_1 and C_2	502	393	129	28
S_4 and C_1 and C_2	367	288	75	29
S_5 and C_1 and C_2	99	96	27	23
S_6 and C_1 and C_2	4	4	4	1
S_7 and C_1 and C_2	431	187	91	28
S_8 and C_1 and C_2	391	257	31	54
Total	3307	2284	700	279

Table 8
Result sets for the performed queries, categorized by publisher.

Query	IEEE	ACM	Elsevier	Springer	Other	Tot.
S_1 and C_1 and C_2	75	4	13	34	51	177
S_2 and C1 and C_2	30	11	23	14	88	166
S_3 and C1 and C_2	29	8	14	14	64	129
S_4 and C1 and C_2	9	2	9	13	42	75
S_5 and C1 and C_2	12	3	1	6	5	27
S_6 and C1 and C_2	1	1	0	0	2	4
S_7 and C1 and C_2	10	4	9	6	62	91
S_8 and C1 and C_2	3	2	6	2	18	31
Total	169	35	75	89	332	700

Literature review

Table 7 outlines the outcome of the review process. This table lists the number of articles that we retrieved by applying criteria IC_1 and IC_2 and the exclusion criteria EC_1 and EC_2. Thereafter, we manually filtered these results by determining whether or not the criteria EC_3 and EC_4 had been fulfilled. This table also shows the number of papers that were the subject of disagreement regarding their inclusion in the set of results. The process of resolving the disagreements is described later in this section. Table 8 shows the result sets for the performed queries, categorized by publisher.

Table 9 illustrates the application of further criteria and the number of articles retrieved. After merging the result obtained from the queries S_1 to S_8, we removed the duplicates, i.e., applied EC_5. The merged result set includes 179 publications on topics related to built-in self tests (BISTs) or software-based self tests (SBSTs) conducted with micro controllers or system on chip (SoC).[8] For example, articles on regression models that are used to estimate stuck-at fault coverage matched at least one of the search queries $S_1 - S_8$ and the context factors $C_{1,2}$. We decided to discard publications on the topics of SBST, BIST, and SoC ($EC_{6,7}$). Many articles could be removed because their focus was not in the area of computer science.[9] By applying criterion EC_8, we excluded all articles that describe AI applications in engineering or math, such as intelligent applications in the domains of medicine, biology, civil engineering, nuclear science, education, and pharmacy. This decisions was made, because these articles do not contribute a research aspect with a strong focus on computer science or software engineering. Some of these articles addressed how to apply AI techniques to improve software/hardware engineering/testing tools instead of the subject of testing anticipatory systems. These articles were subsequently removed from the set of results (EC_9). Finally, we applied EC_{10} and excluded articles in which an SuT exhibiting at least one the key stages of anticipatory systems was not investigated (see Table 5 and Fig. 1). For this purpose, we manually selected publications that mentioned at least one of the four key stages of anticipatory application.

This selection process was carried out by following a rigorous procedure and relied on the abstract and – where necessary – the full text of the publication. This selection process was conducted by two authors. If both agreed that the article should be directly included or excluded, the respective decision was made. For those articles where immediate agreement was not reached, we led a workshop to resolve the disagreement. As part of this process, we discussed Table 14 in the Appendix, which lists the obtained set comprising 61 articles.

Snowballing

Snowballing refers to the process of using the reference list of an article (backward snowballing) and the paper citations (forward snowballing) to identify additional articles. We conducted

[8] A system on a chip (SoC) is an integrated circuit that integrates all components of a computer or other electronic systems. It may contain digital, analog, mixed-signal, and often radio-frequency functions on a single substrate.

[9] Note that EC_2 excludes only articles that are not in the field of computer science, math, or engineering.

Table 9
Filter criteria to obtain our set of results.

Criteria	No. publications	Disagreement
EC_5	696	0
EC_6	639	21
EC_7	476	26
EC_8	259	38
EC_9	91	29
EC_{10}	61	5

Table 10
Results from the forward and backward snowball sampling.

Iteration no.	Art. no. fwd.	Art. no. backw.	Tot. w. dup.	Tot. wo. dup.
1	18	19	37	24
2	17	35	52	34
3	25	21	46	31
4	5	27	32	25
5	3	21	24	17
6	1	6	7	7
7	2	0	2	2
Total	71	129	200	145

snowball sampling using the obtained set of results as a seed set (initial set). We followed the guidelines for snowballing provided in the systematic literature reviews outlined in Wohlin (2014).

Seed set

The seed set included papers from the following publishers: the Association for IEEE Computer Society, Springer Verlag, the International Foundation for Autonomous Agents and Multiagent Systems, Computing Machinery, Elsevier, IOS Press, and SAE. Among the main contributors to the seed set are researchers in the adaptive and self-managing systems community and the AI and software engineering community. Furthermore, the researchers in scientific communities focusing on simulation and autonomous agents contributed to our seed set. Most frequently the authors of the articles in the seed set tagged their articles with the following keywords: software testing, adaptive systems, embedded systems, runtime, software engineering, computer software, regression testing, testing, autonomous agents, changing environment, cyber–physical systems, robots, software testing, and software reliability.

Backward and forward iterations

Starting with our seed set comprising 61 articles, we conducted seven snowballing iterations. After the seventh iteration, no new articles were found, and the snowballing procedure was concluded. Table 10 lists the number of publications identified at every forward and backward iteration step. A substantial increase in the number of identified articles in a subsequent iteration might indicate the presence of a possibly undiscovered cluster (Wohlin, 2014). As outlined in Table 10, iteration no. 2 enabled us to potentially reach a sufficiently good set that covered the major clusters with respect to our research questions, as the number of identified publications (after eliminating duplicates) decreased with each subsequent iteration. Ultimately, we retrieved 206 articles: the 61 articles from our seed set (database search) and the 145 articles obtained through the snowballing procedure.

3.1. Classification procedure

We carried out the analysis and classification procedure with the obtained set of results. We classified each article in this set by its research, contribution, and system type. Thereby, a single article may contribute to several categories. For example, an article may suggest a solution (research type: solution proposal) and present an evaluation of this solution (research type: evaluation research). The article may suggest a method (contribution type: framework/method) in combination with a formal model (contribution type: model). The investigated SuT might be an ADAS function. The article might focus on testing the collection of data from sensors and on testing the feature extraction from this raw data (system types: context sensing and inferring context). We classified the result set according to research type, contribution type and system type. Regarding the research type, we disagreed on the classification of 22 articles (10 percent). The disagreement rate with respect to the contribution type resulted in a similar rate (26 articles, 13 percent). In 38 cases (18 percent), the classification according to the system type resulted in disagreements. We led a workshop to resolve these disagreements. With respect to the system type, we applied the developed guidelines G_1 to G_4 to reach a consensus.

We provide a Microsoft Excel Sheet that covers the final result set of publications and the classifications, accessible at https://bit.ly/2WfcWqq.[10]

4. Results

First, we provide an overview of the obtained results and then we address RQ 1: the general pool of publications, RQ 2: how the contributions can be quantitatively characterized, and RQ 3: the observed trends regarding the testing of anticipatory applications.

Fig. 2 outlines the number of publications over time. Although the first publications appeared in the early 1990s, the figure shows that a publications continuously appear starting in 2004. From this point in time, the number of articles on testing anticipatory applications increases considerably. Regarding the research type, most contributions address solution proposals or philosophical papers. In recent years, most of these articles have contributed either to the topical area of frameworks/methods or suggested models for testing. Given the general trend, the figures illustrate that the number of solution proposals has grown more strongly than this trend. Furthermore, as Fig. 2 shows, the number of articles on frameworks/methods outperforms the general trend. Recently, the number of articles on models in testing increased noticeably.

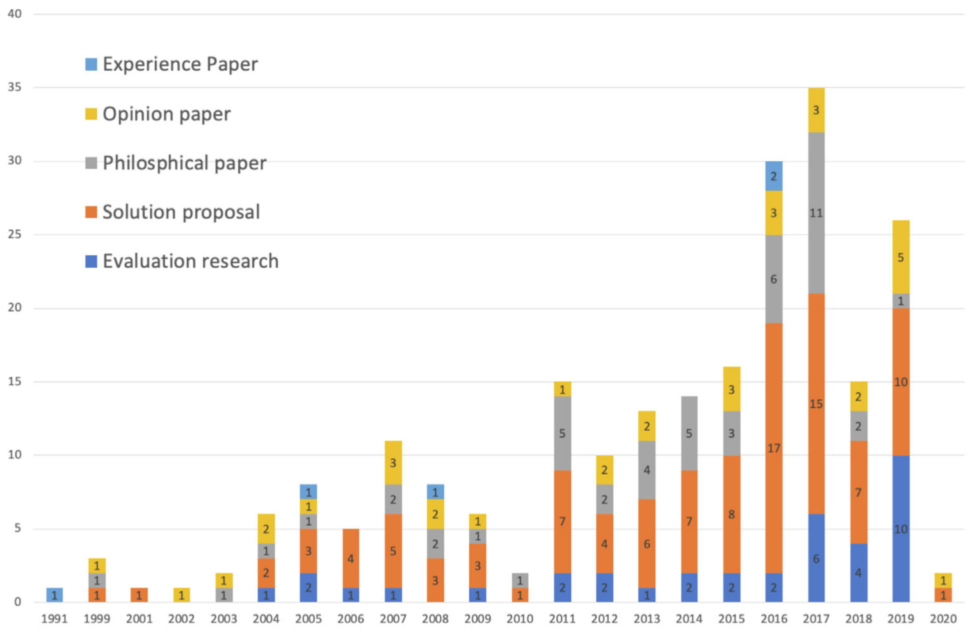
RQ 1: General pool of publications

To provide an overview of the identified articles, we categorized the result set by research type (see Table 3), contribution type (see Table 4), and system type (see Table 5). Fig. 2 provides a consolidated picture, showing the overall number of publications regarding the research and contribution types.

Regarding the research type, Fig. 2 indicates the presence of a trend towards solution proposals. Out of the 206 papers in the result set, we classified 30 contributions as opinion papers and 52 as philosophical articles. We classified 26 articles as making a contribution to evaluation research. Taking into account the general trend over time (see Fig. 2), the classification according to the research type strongly indicates that this is a young and evolving field of research: Almost 62 percent of the articles propose new solutions (solution proposals) or discuss different viewpoints (opinion papers).

Fig. 3 presents a systematic map that correlates the research and contribution types. Regarding the contribution type, 136 articles suggest that frameworks/methods should be used to test anticipatory applications. Most of these articles suggest using solution proposals or models. Correspondingly, evaluation research was primarily conducted by evaluating frameworks and models.

[10] Last accessed 9th Sep. 2021.



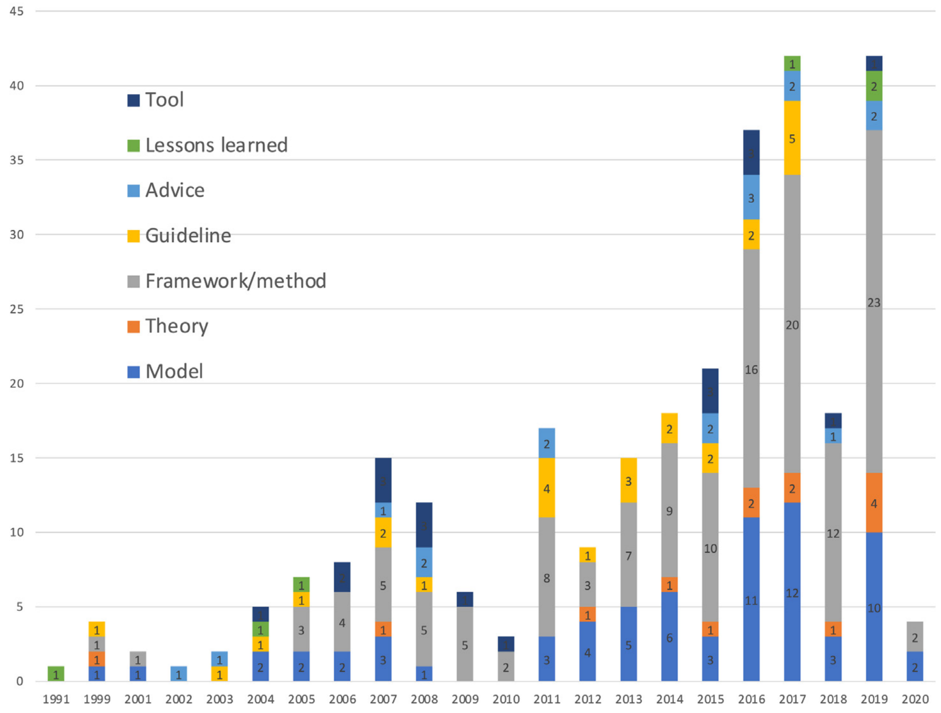


Fig. 2. Overview of the absolute numbers of publications and their distribution. The number of solution proposals and the number of frameworks/methods outperforms the general trend in testing anticipatory applications.

However, the number of evaluations appears to be smaller than the number of solution proposals. Fourteen (14) articles primarily contribute to theory, and only six articles report on lessons learned. The strong trend to use framework/methods to test anticipatory applications and the small percentage of theoretical papers further indicates that this is a young but evolving field.

RQ 2: result set contributions

While classifying the result set, we collected information on the SuT. We assigned four possible system types to the 206 papers in the result set: context sensing, inferring context, predicting context, and intelligent actioning.

Coverage of key stages

Fig. 4 illustrates the distribution of the system type with respect to the contribution and research types. The four key stages are primarily addressed in terms of solution proposals and philosophical papers that contribute by providing frameworks/methods and models. For every contribution type, the number of articles on intelligent actioning exceeds the number of articles categorized in the remaining stages. The interest in framework/methods is higher than that in models in all phases except for the context prediction stage. Evaluation research plays a smaller role but, nonetheless, all four stages are addressed. Because of the limited number of publications reporting experiences, offering advice for practitioners, or sharing lessons learned,

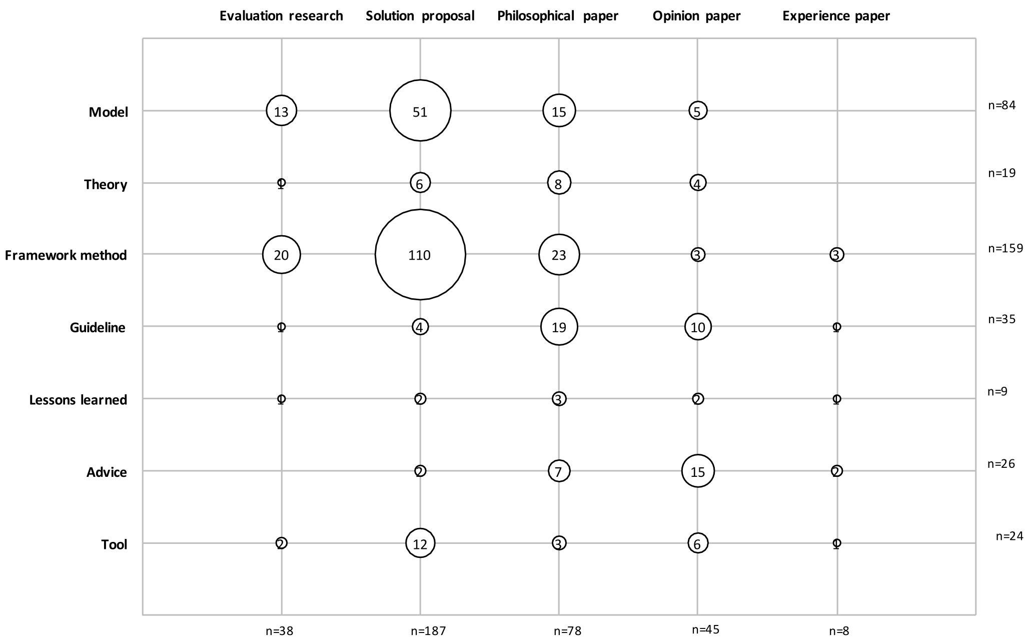


Fig. 3. Systematic map of research and contribution types.

we conclude that empirical research plays a minor role in testing anticipatory systems. Furthermore, theoretical contributions are underrepresented. In our result set, all key stages are addressed. This indicates that the published papers as a whole (unlike the individual contributions) address the testing of anticipatory systems, comprising the full spectrum of the identified stages.

Venue information

Table 11 lists the most common peer-reviewed venues (conferences, symposia and workshops) extracted from our results set. Six publications were published at the IEEE Intelligent Vehicles Symposium, and five contributions in our result set appeared at the International Joint Conference on Autonomous Agents, Multiagent Systems (AAMAS) and the International Conference on Artificial Intelligence Testing (AITest). Authors attending the IEEE Aerospace Conference, the ACM Symposium on Applied Computing, and the IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop each contributed three papers to our result set. Each of the following conferences were attended by authors contributing two articles: the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), the International Computer Software and Applications Conference, the International Conference on Software Engineering (ICSE), and the IEEE International Symposium on High Assurance Systems Engineering.

Scientific, peer-reviewed conferences, and journals

Four articles in our result set appeared in the Journal of Systems and Software followed by the Journal of Autonomous Agents and Multi-Agent Systems and the Journal of Advanced Robotics (each with three articles). IEEE Transactions of Software Engineering, Artificial Intelligence Review, the Software Quality Journal, and the Journal of Information and Software Technology each contributed two publications to our result set (see Table 12).

RQ 3: trends in testing anticipatory applications.

Overview of solution proposals

Among the 126 solution proposals, we identified trends. To do so, we studied these articles and classified all solution proposals by the proposed approach. For each trend, we selected a number of articles. Table 13 lists the identified trends and the system type assigned. In the following section, we briefly discuss these trends. In particular, we provide arguments for the interpretation of context and the system type being addressed.

Online monitoring. Gupta and Schumann (2004) point out that work on development of neural networks and adaptive technology itself and on its V&V methods is crucial for NASA missions, as well as beneficial to numerous other agencies and the aerospace industry. The authors propose using a method to analyze the performance of a neural network during V&V and during its operation as a monitoring device. Their approach suggests that, for control applications, it is important that the controller's model of the system reflects the actual system behavior with sufficient accuracy. This means that it needs to be shown that the modeling error does not exceed a given threshold for the entire operation envelope. To measure the reliability, the neural network output is of great interest. The article's authors suggest placing a statistical error bar on the output of a neural network to predict a probability regarding how close the estimated and the actual values are. The error bar serves as a prediction for the range of the model output, whereby the actual value is a probability of more than 95 percent. By assuming a normal error distribution, confidence intervals can be related to the standard deviation in the network output (Gupta and Schumann, 2004). Regarding this solution proposal, the SuT is an Intelligent Flight Control System that was developed at NASA Dryden. The adaptive control architecture uses a proportional–integral–derivative controller and a neural network. This system covers all four key stages of anticipatory systems. The context in this case refers to environmental conditions that may arise during a mission.

Haydarlou et al. (2007) proposes taking a model-based approach to self-monitoring, describing structural and behavioral models of a system at different levels: application, subsystem, component, and class level. In their approach, a system's behavior is monitored in the context of a hierarchy of use cases related to these levels. The structural and behavioral models are used to automatically instrument an existing distributed system. The authors illustrate their approach with a secure portal application in the area of banking. Data collection, extraction of conditions

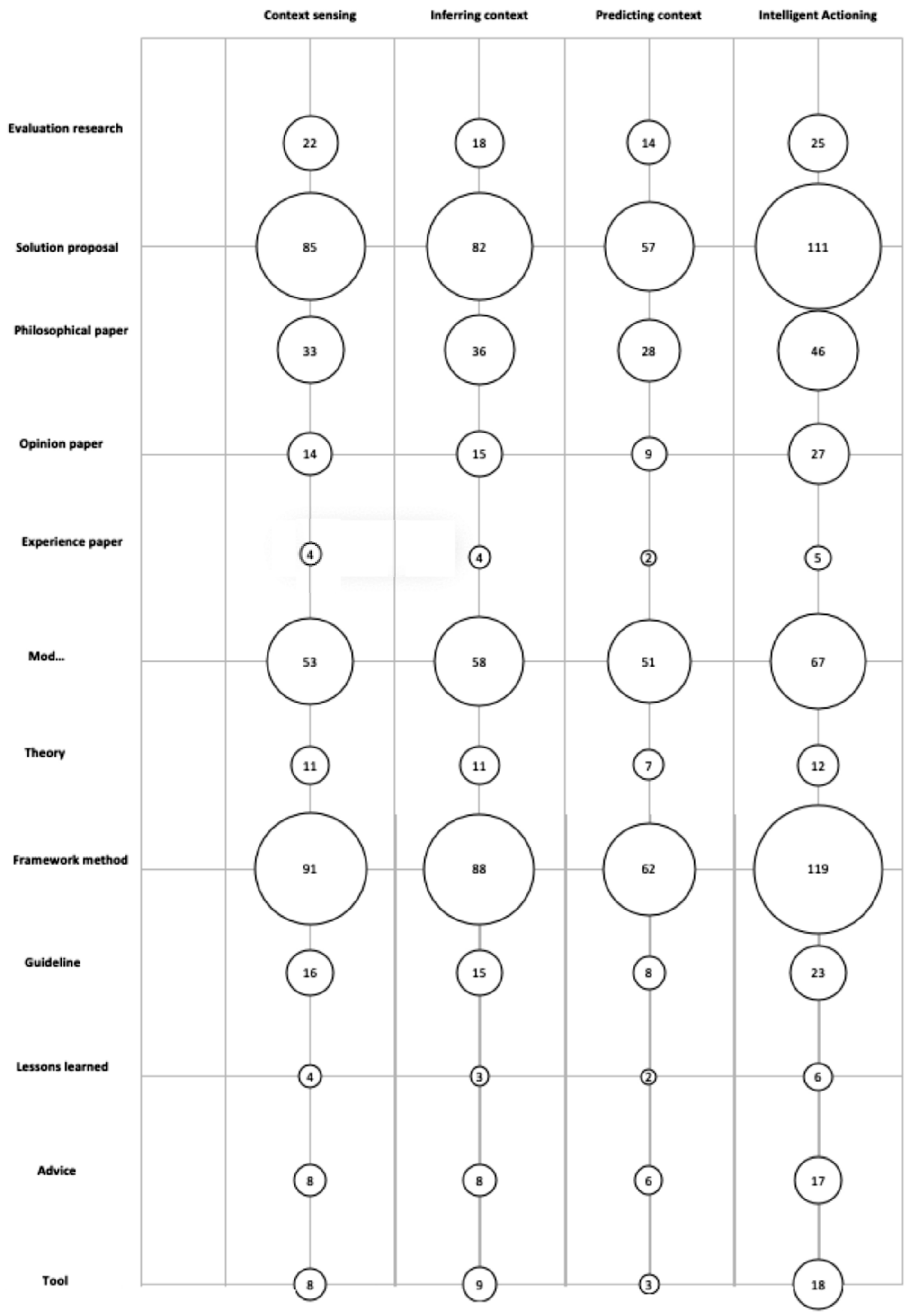


Fig. 4. The distribution of research and contribution types with respect to system types.

from the collected data, as well as intelligent actioning is covered, but the authors do not clearly define the term context.

Goldsby et al. (2008) propose using AMOEBA-RT, a runtime monitoring and verification technique to ensure that dynamically adaptive software fulfills its requirements. This is achieved by expressing adaption properties in A-LTL, an adaptive variant of linear-temporal logic (LTL, Zhang and Cheng (2006)) and LTL. AMOEBA-RT instruments the adaptive system to achieve runtime monitoring. This instrumented code collects information about the runtime state of the adaptive software and transmits the information to a runtime model checking server. The instrumentation is achieved using AspectJ, an aspect-oriented extension of the Java programming language. For runtime monitoring, the AMOEBA-RT approach is illustrated with a multi-threaded Java program, whereby data are processed and transmitted from one thread to another in a pipelined fashion. The authors implemented an adaption mechanism for the used pipeline to monitor the CPU workload and enable a decision-maker to select the optimal implementation for specific run-time conditions. Here, context refers to the computing and communication infrastructure.

With respect to monitoring, Taylor and Marjorie A. Darrah (2003) mention various techniques, such as data sniffing, safety

Table 11
Distribution of papers at peer-reviewed venues and the addressed key stages: S - context sensing, I - inferring context, P - predicting context, A - intelligent actioning.

Venue	Key stage				Authors (year)
	S	I	P	A	
IEEE Intelligent Vehicles Symposium				x	Gruyer et al. (2014)
	x			x	Zofka et al. (2014)
	x	x	x	x	Bach et al. (2016)
	x			x	Zofka et al. (2016)
				x	Bosse et al. (2006)
International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)				x	Nguyen et al. (2008b)
				x	Nguyen et al. (2008a)
		x		x	Zhang et al. (2008)
IEEE Aerospace Conference	x		x	x	Schumann et al. (2005)
	x		x	x	Schumann and Liu (2007)
		x			Visnevski and Castillo-Effen (2010)
ACM Symposium on Applied Computing				x	Lee et al. (2017a)
	x		x		da Costa et al. (2010)
AIAA/IEEE Digital Avionics Systems Conference	x	x	x	x	Broderick (2004)
	x	x	x	x	Broderick (2005a)
IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, (SASOW)	x			x	Jahan et al. (2017)
		x		x	Hänsel et al. (2015)
				x	Markov and Fröhlich (2015)
IEEE International Conference on Software Quality, Reliability and Security	x	x		x	Moran et al. (2017)
	x	x	x	x	Reichstaller and Knapp (2017)
	x	x			Eberhardinger et al. (2016a)
International Symposium on Software Engineering for Adaptive and Self-Managing Systems, (SEAMS)		x		x	Fredericks and Cheng (2015a)
		x	x	x	Mongiello et al. (2015)
International Computer Software and Applications Conference				x	Lee et al. (2017a)
	x	x	x	x	Lim et al. (2015)
	x			x	Wang and Zhu (2012)
International Conference on Software Engineering				x	Coelho et al. (2006)
		x		x	Bai et al. (2011)
IEEE International Symposium on High Assurance Systems Engineering	x			x	Andrews et al. (2016)
	x	x	x	x	Gupta and Schumann (2004)
IEEE International Conference On Artificial Intelligence Testing (AITest)			x	x	Bozic et al. (2019)
			x	x	Guichard et al. (2019)
	x	x	x	x	Byun et al. (2019)
				x	Marijan et al. (2019)
	x				Zhu et al. (2019)

Table 12
Distribution of papers in peer-reviewed journals and the addressed key stages: S - context sensing, I - inferring context, P - predicting context, A - intelligent actioning.

Journal	Key stage				Authors (year)
	S	I	P	A	
Journal of Systems and Software	x	x	x	x	Qin et al. (2016)
	x				Mattos et al. (2017a)
	x	x		x	Yang et al. (2018)
	x	x		x	Liu et al. (2006)
Autonomous Agents and Multi-Agent Systems		x		x	Low et al. (1999)
	x	x	x	x	Winikoff (2017)
		x		x	Nguyen et al. (2012)
Advanced Robotics	x	x	x	x	Abdelgawad et al. (2017)
			x		Wienke and Wrede (2017a)
Artificial Intelligence Review	x	x	x	x	Li et al. (2018a)
	x	x	x	x	Hernández-Orallo, J. (2017)
IEEE Transactions on Software Engineering	x	x	x	x	Calinescu et al. (2017b)
	x	x	x	x	Padgham et al. (2013)
Information and Software Technology	x	x	x	x	Lee et al. (2018a)
	x	x	x	x	de Sousa Santos et al. (2017)
Software Quality Journal	x	x	x	x	Liu et al. (2007)
	x	x	x	x	Patel and Hierons (2017)

Table 13
Trends in testing anticipatory applications and the addressed key stages: S - context sensing, I - inferring context, P - predicting context, A - intelligent actioning.

Trend	Key stage				Authors (year)
	S	I	P	A	
online monitoring	x	x	x	x	Gupta and Schumann (2004)
	x	x	x	x	Haydarlou et al. (2007)
	x	x		x	Goldsby et al. (2008)
test-process lifecycle	x		x		Geddes (1991)
		x		x	Chander et al. (1999)
	x	x	x	x	Schumann et al. (2005)
frameworks for agent testing		x		x	Low et al. (1999)
		x	x	x	Bai et al. (2011)
		x	x	x	da Costa et al. (2010)
		x	x	x	Zhang et al. (2011, 2008)
runtime V&V		x		x	King et al. (2011)
		x		x	Mongiello et al. (2015)
		x		x	Sharifloo and Spoletini (2013)
		x		x	Fredericks and Cheng (2015a)
		x		x	Lee et al. (2018a)
model-based and simulation-based testing	x	x	x	x	Andrews et al. (2016)
	x	x	x	x	Abdelgawad et al. (2017)
		x	x	x	Eberhardinger et al. (2016b)
	x	x	x	x	Saglietti et al. (2015a)
	x	x	x	x	Saglietti and Meitner (2016)
continuous exp. and feedback loops	x			x	Bosch and Olsson (2016c)
	x				Mattos et al. (2017a)
	x				Olsson and Bosch (2014)
	x				Bosch and Eklund (2012a)
quantitative verification	x	x		x	Gerasimou et al. (2014)
	x	x	x	x	Calinescu et al. (2017b)
	x		x	x	Calinescu et al. (2017a)

monitors, and the use of Lyapunov stability analysis (Feyzmahdavian et al., 2018) at runtime. The discussion of the paper focuses on the verification and validation of neural networks, and the analyzed systems are assumed to cover all of the different stages of anticipatory systems. The issue of context is not explicitly addressed by the authors.

Testing process and system life cycle. Already in 1991, the authors of Geddes (1991) proposed a procedure for the V&V of the Pilot's Associate, one of the most comprehensive and complex real-time artificial intelligence decision aids at that time. The authors note the prototyping development process results in changes in both the requirements and actual implemented functionality over the course of a prototyping cycle. For this reason, they argue that a close coordination between the test planning process and the development process is necessary. The proposed test process involves mission environment coordination, a synthesis of requirements, test planning, and test environment coordination. The SuT acquires sensory data (sensing stage) and provides recommendations for the pilot (context prediction). The concept of context covers issues such as thread positions, threat behavior, mission goals, and rules of engagement.

The authors of Chander et al. (1999) propose a three-tiered life-cycle model that can be used to integrate V&V into a rule-based system life cycle. As rule bases evolve due to user feedback in response to test operations, test results, or changing environments or requirements, the authors suggest applying an incremental V&V process. They conclude that the system structure should play a major role in system evaluation and that an incremental V&V process may reduce costs and effort without compromising the quality of testing such systems. The authors assume that certain requirements and a structure for the rule-based systems exist and do not focus on context or environment. They refer to the idea of context as unanticipated problems.

The authors of Schumann et al. (2005) propose taking a three-layered approach to V&V techniques for intelligent adaptive controllers. Their solution proposal involves using a core group of rigorous techniques, such as stability testing and Vapnik–Chervonenkis dimension arguments (Péez-Sánchez et al., 2014), to determine the potential of learnability. The goal of using such methods is to provide asymptotic guarantees. Because this is not sufficient to guarantee properties such as reliability or safety, the authors suggest performing testing and checking for errors in the second layer. According to Schumann et al. (2005), testing an adaptive controller, however, is a far from trivial process, because testing needs to take into account the system's history and the current state of the adaptive component. The authors note that V&V tools will need to be designed and fine-tuned to meet these demands; however, the issue of how to achieve this remains open. For "truly"(we interpret this as covering all four stages of anticipatory systems) adaptive systems, Schumann et al. (2005) suggest a third layer. This layer monitors how the actual system behaves and retrieves information about whether the state of the system is recoverable.

Testing agents. Bai et al. (2011) point out that systems relying on the service-oriented architecture (SOA) paradigm impose new requirements on automatic and collaborative testing. As services are dynamically reconfigured and recomposed, the testing system needs to be made aware of changes in the SuT, re-select test cases, and re-schedule test plans in accordance. The authors developed an agent-oriented approach to perform coverage-based testing and performance testing. First, this contribution proposes a design for a test agent model that can support autonomous and cooperative testing of services in a distributed environment. Second, the authors suggest that agent intelligence can be improved with sophisticated designed knowledge and rules, so that they can improve test efficiency and reduce test costs. The SuT is a service deployed at a Tomcat server, and the paper primarily addresses all the key stages except sensing. Context in this case refers to computing resources and the user type.

da Costa et al. (2010) highlight the fact that self-adaptive software systems can check the needs for the adaptations, perform them, and ensure their compliance with new environment requirements. Therefore, their approach provides an extension to the Java self-Adaptive Agent Framework (JAAF) in order to apply the self-test concept. This framework allows for the creation of self-adaptive agents based on a process composed of a set of four main activities (monitor, analyze, plan, and execute). da Costa et al. (2010) propose extending the process and framework by including test activity that will check the adapted behavior before its execution. The SuT exhibits behavior covering all key stages except sensing. These authors define context as any unpredictable change forcing the agent to self adapt.

The authors of Gücan and Bernon (2013) present the design of a novel generic framework for testing agent-based simulation models. They identify two basic elements for testing: agents and simulated environments. To test each use case for these elements, their approach can be used to develop a test scenario specifically for the considered use case. For each test scenario, a special agent observes the model elements under test and executes tests to determine whether these elements conform to the desired behavior. Context primarily refers to the environment that can be simulated and the data these environments use or produce. As in the previously discussed papers on agents, most stages of anticipatory systems are addressed; however, the focus on the sensing stage is weaker.

The authors of Low et al. (1999) suggest using certain methods to automatically generate test cases for multi-agent systems based on the BDI (beliefs, desires, and intentions) architecture. BDI agents have beliefs about themselves and the surrounding environment. Each agent has its own desires or goals in its execution life. These desires may be in the form of reactions or long-term goals. Reactions can be caused by changing beliefs or responses to other agents. These reactions usually form short-term goals. Agents can also have long-term goals that can exist as long as the agents exist. The agents strive to fulfill their short- and long-term desires by invoking plans (Low et al., 1999). The authors employ two criteria to steer the generation of test cases, namely, the coverage of plans by BDI agents and coverage of nodes within plans. The authors apply the proposed test generation technique to three systems. For the purpose of discussion, they use a web-crawling system called BibSearch to search for full bibliographic references in the World Wide Web. Due to the planning component, the aspect of inferring and predicting the context is covered, and the agent is built to conduct some kind of intelligent actioning. The idea of context here refers primarily to conditions for entities (e.g., in plans).

The authors of Zhang et al. (2011, 2008) propose a process and method of testing agents and show how to generate unit tests on a goal–plan agent system. They focus on unit testing and identifying the appropriate units, as well as present mechanisms for generating suitable test cases and for determining the order in which the units of the intelligent agents are to be tested. They argue that the suggested approach, concept, algorithm, and process can be adopted to any plan and event-based agent system. The authors use an Electronic Bookstore system, an agent-based system dealing with online book trading, which contains agents that perform certain functions such as handling customer relations, a delivery manager, a sales assistant, and a stock manager agent. The stock manager agent was considered the SuT, and the overall system needs to infer context, apply reasoning to predict possible future states, and carry out some intelligent actioning that might influence the system itself. The idea of context refers to the conditions of the abstract entities, such as the agent's plans and beliefs.

Run-time V&V frameworks. King et al. (2011) suggest using self-testable autonomic components (STACs). STACs encapsulate the dynamically replaceable baseline components of the system, which represent the software services being managed (i.e., components are services such as a local service or a web service). If an autonomic change replaces a baseline component with a delta component that has never been used in the context of the application, self-testing is performed as part of its integration into the system. In the proposed approach, an autonomic manager continuously monitors the system environment. It responds to environmental changes, selects the components that need to be replaced, and dynamically configures the system. A test manager intercepts this adaption and first generates a test plan, then selects the stubs that may be required for testing and passes them on to the autonomic manager to set up the validation process. Subsequently, the test manager sets up a sequence of a series of self-tests using the STACs and evaluates the results. In this way, the system can determine whether the adaptive change is accepted or rejected. The presented case studies, such as an autonomic container or autonomic job scheduler, are considered as anticipatory systems, as these applications respond to changes in their environment by inferring the context and taking intelligent actions. However, the authors of King et al. (2011) focus on considerations of how adaption will affect the internal structure and behavior of system components.

Complementary to this approach is the one suggested in Munoz and Baudry (2009) that addresses the external aspects of testing adaptive systems. Their work on Artificial Shaking Table Testing (Munoz and Baudry, 2009) involves investigating whether or not an adaptive system responds correctly to environmental changes. Using this technique, the authors select test data that simulate environmental changes called context-shakes.

Sharifloo and Spoletini (2013) propose LOVER, a method for lightweight formal verification of adaptive systems at runtime. Unlike classical model checking, LOVER can deal with incomplete models, whereby a number of components are unspecified at design time. LOVER's two-phase approach indicates that it can be used to overcome this limitation. In the first phase, the designer verifies the incomplete system specification at design time and generates a set of constraints for the unspecified components. These constraints are verified at runtime whenever the specification of the components becomes available. However, the authors mention that the time and space required for the verification could be considerable. Since some bindings are resolved only while the system is operating, the total overhead in resolving them should also be kept as small as possible. Context inference and intelligent actioning refer to the ability of the system to integrate a component when its specification becomes available, infer the assumptions regarding the environments and subsystems, and to perform the run-time verification task. Context thus refers to the behavior of some components and the environments encountered at runtime.

With the Proteus, a requirement-driven approach for managing runtime testing, Fredericks and Cheng (2015a) suggest an approach that can be taken to perform testing activities at runtime, including test execution and online adaptation. Proteus provides two levels of test adaptation: test suite adaptation and test case parameter value adaptation. Within Proteus, an adaptive test plan can be used to configure the self adaptive system at design time, whereby each configuration corresponds to a particular operating context, and each adaptive test plan may comprise a default test suite and a set of automatically derived test suites. By taking the Proteus approach, default test suites are derived at design time, and intermediate test suites are derived at runtime. The authors suggest defining an adaptive test plan for each operating context, whereby context refers to a configuration of system and environmental parameters. Based on this context, the system can choose the given context and conduct intelligent actions.

Mongiello et al. (2015) propose using AC-contract, an approach to runtime verification that covers the modeling and verification of adaptable software systems. Their technique relies on design-by-contract and a reflection mechanism for implementing adaptability. Starting from system-level requirements, the authors show how to identify properties that apply to single components. These properties are then represented as contracts in the programming language. The contract checker implements the mechanism for searching for the adaptable components that verify the contract specifications.

Lee et al. (2018a) propose using RINGA, a self-adaptive software framework employing finite state machines for design and runtime verification. The design-time part of RINGA is responsible for designing a finite state machine model (the software model) and extracting data to be monitored during the runtime phase. This finite machine is evaluated to check that the model satisfies all requirements. After the evaluation of this software model, monitoring data and triggers are extracted. Monitoring data are related to the environment or software changes that occur during software implementation, and triggers are factors for changing the software. After extracting the monitoring data and triggers, the state machine model is abstracted into a simplified finite state machine and used for runtime verification. For this purpose, the authors propose using an abstraction algorithm, which enables a model checker to cope with the complexity of the abstracted finite state machine. The SuT is the prototype system Traveller, an app supporting travelers before a trip and during a trip, helping them to plan a journey. The user interaction is subject to variations and evolution and depends on changes in the context. The application adapts to the contextual changes which cannot be controlled by the user with a set of functionalities that are dynamically loaded and executed, i.e., it infers the context and performs intelligent actions.

Model-based and simulation-based testing. Andrews et al. (2016) and Abdelgawad et al. (2017) propose using a world model to test autonomous systems using Petri Nets. They show how to apply a systematic model-based test generation approach to generate test cases from a world model of an autonomous system. The authors suggest taking a three-phase approach. First, they construct a structural model of actors and thereby capture their attributes, functions, and relations. Second, the authors construct a behavioral model to describe the possible states and transitions for each of the actors. In phase 2, they apply graph-based coverage criteria to generate the test paths and an input-space partitioning coverage criterion to generate test data. In phase 3, the test paths and test data are first transformed into executable test cases. The SuT is an autonomous ground vehicle consisting of four modules: world perception and modeling (context sensing), localization and map building (context inference, context prediction), and path planning and decision-making (intelligent actioning).

Eberhardinger et al. (2016b) present an approach for back-to-back testing of feedback-loop-based self-organization mechanisms. With their model-based approach, the authors address three challenges faced in back-to-back testing: supplying test oracles, systematic test case selection, and automation of test execution. Their test model mainly consists of two parts. First, the model of the system controlled by the self-organizing mechanism, i.e., the environment model, and second, the model of the intended behavior of the self-organizing mechanism, i.e., the test model. The intended behavior of the system is described as a set of constraints capturing a corridor of correct behaviors. These constraints are then used to implement the test oracle that evaluates these constraints on the actual state of the system. The SuT is a future production scenario in the form of a self-organizing production cell. Such a cyber–physical system will integrate self-organizing mechanisms to decentralize decision-making and to react on component failures at runtime (intelligent actioning). The production cell is able to reorganize the carts and robots (context sensing, context inference) and has the ability to change tools whenever necessary. With their idea of context, the authors refer to changes in the system and in the environment that occur at runtime. This makes it impossible to predetermine every possible future system state at design time.

Saglietti and Meitner (Saglietti et al., 2015a; Saglietti and Meitner, 2016) review several model-based testing approaches for autonomous, cooperative, and reconfigurable robots. All testing strategies presented have similar underlying modeling notations, namely, all of the discussed approaches use Colored Petri Nets. According to the authors, this modeling paradigm offers multiple advantages by supporting a compact and scalable representation of complex behavior. The authors note that, in addition to regular behavior, autonomous robot decisions need to be taken into account, such as the occurrence of anomalies by reconfiguration strategies. Saglietti et al. (2015a) discuss the processing stages of a robot covering sensing, perception, reasoning, and action and extend this to cooperative robots.

Continuous experimentation and feedback loops. Despite the fact that agile development methods were adopted in the last decade, involving stakeholders to collect feedback is still a crucial and complex task. The requirements regarding the software product are usually collected, prioritized, and broken down into digestible tasks. The process of eliciting requirements – although it has been studied exhaustively – is still an opinion-based process typically led by a senior product owner (Bosch, 2012). AI-enabled systems anticipate future situations and adapt their behavior dynamically. Thus, the uncertainty in the behavior of the SuT is not only rooted in this opinion-based process, but also in the adaptive nature of the system.

In order to close the loops between the development team, the system, and the various stakeholders, Olsson and Bosch (2014) suggest adopting experiment-driven development practices, following the idea of the minimal viable product (Lenarduzzi and Taibi, 2016). For example, at the feature level, the authors of Kohavi et al. (2012) report on the successful application of controlled experiments (specifically A/B-testing) to make data-driven decisions. In this case, the SuT is deployed to a production environment, and the software, the product, or service is instrumented to collect metrics about what version customers appreciate most (context sensing). In this process, the collected data are used to evolve requirements. Olsson and Bosch (2014) report on two specific use cases that describe how to close this loop from requirements to testing. With the IoT, it is possible to deploy software for virtually all devices. To this end, the authors of Bosch and Eklund (2012a), Bosch and Olsson (2016c), Mattos et al. (2017a), Olsson and Bosch (2014) and Kohavi et al. (2012) report on setting up experiment-driven software engineering practices in the context of eternal embedded systems.

Mattos et al. (2017a) propose different design solutions for a framework that supports automated experimentation. The discussed architecture is based on decisions, evaluations, and experiences gained in different industries. The authors suggest taking an approach that considers manual as well as automated experimentation. The authors illustrate their proposal in terms of a human–robot interaction problem.

These approaches to automated experimentation strive to explore the behavior of the system within an environment that is as close as possible to the production environment. Context refers to configurations and parameters causing behavior that are outside the initial experiment boundaries.

Quantitative verification. Quantitative verification is a technique used to analyze quality attributes such as the correctness, performance, or reliability of systems exhibiting stochastic behavior. States represent the various system configurations, whereas the transitions between these states are annotated with probabilities. Given models such as discrete-time Markov chains, Markov decision processes, and a property specified formally in a variant of temporal logic that is extended with probabilities, quantitative verification is used to analyze the model and to decide whether or not the property applies to the model. With the help of a probabilistic model checkers like PRISM (Kwiatkowska et al., 2011), this can be performed entirely automatically.

Calinescu et al. (2017a) suggest applying runtime quantitative verification (RQV) to provide evidence that the self-adaptive software meets its dependability, performance, and other QoS (Quality of Service) requirements. The article points out that RQV requires continual learning and the quantitative verification of probabilistic models. In particular, the authors address the need to perform runtime learning and update probabilistic models on the basis of observations on the behavior of the modeled software. The authors use the continual verification framework COVE (Calinescu et al., 2013) for the RQV-driven verification of a service-based system. These services are deployed in a cloud infrastructure and thus may benefit from the elastic nature of the cloud. Adaptions may happen due to the need to increase service availability or to react to unexpected changes (intelligent actioning), such as physical server failure or the consequences of software faults (context sensing, context inference).

The authors of Calinescu et al. (2017b) suggest using ENTRUST - a methodology that can be applied to perform systematic engineering on trustworthy self-adaptive software. ENTRUST addresses the most common type of control loops used to build up self-adaptive systems, following the monitor–analyze–plan–execute (MAPE) principle, thus covering all four key stages of anticipatory systems. The ENTRUST methodology can be used with different combinations of modeling-, verification-, and controller-enactment methods. Thus, different self-adaptive system architectures and assurance and verification tasks may result. In Calinescu et al. (2017b), the authors present an ENTRUST instance where a reusable verified controller platform is present in the core of the suggested architecture. A trusted virtual machine interprets and executes models of the MAPE control loop, and a probabilistic verification engine is used to verify stochastic models of the controlled system and its environment. This engine consists of verification libraries and the probabilistic model checker PRISM. As such, this ENTRUST instance works with a stochastic finite state transition model of the controlled system and the environment. Properties are expressed in probabilistic temporal logic, e.g., probabilistic computation tree logic when PRISM is used with continuous-time Markov chains. According to Calinescu et al. (2017b), this use of the ENTRUST methodology is applicable to self-adaptive systems whose requirements can be specified in probabilistic temporal logic and whose behavior can be described using stochastic models. In Calinescu et al. (2017b), two adaptive SuT are mentioned. First, the authors apply ENTRUST to an unmanned underwater vehicle and second, to a foreign exchange trading system.

Gerasimou et al. (2014) extend conventional software engineering methods, adapting them to RQV, and suggest three techniques, namely, caching, lookahead, and nearly-optimal reconfiguration to improve the response time of RQV. For example, by caching the recent verification steps, the authors reuse such techniques and thus can reduce the number of runtime re-verification steps when the system changes are small and localized. Furthermore, the authors suggest using idle CPU cycles to pre-verify states that are likely to occur in the near future. The techniques are applied to an unmanned underwater vehicle exhibiting adaptive behavior and cover the stages of context sensing, context inference, and intelligent actioning. The concept of context refers to incomplete models that include unknown or estimated parameters, non-determinism (i.e., alternative options whose likelihoods are unknown), parts that are missing, or some combination of all of these (Gerasimou et al., 2014).

5. Discussion of mapping study results

Given the four key stages of anticipatory systems and the role of context, a life cycle for managing the context might be desirable. Such a life cycle would address tasks such as defining and sensing the context, handling and evolving the context, provisioning, maintaining and disposal (Tamura et al., 2010). When developing hypotheses about such a life cycle, we must consider that the identified trends contribute to mastering this life cycle and cover the identified system type facets differently.

To achieve anticipatory behavior, these systems must be able to monitor the changes with executable models that can detect changes in context variables that might result in intelligent actions. The solution proposals to this range from statistical models to structural and behavioral models and on to the specification of adaption properties in LTL. Adaption properties themselves address non-functional qualities such as stability, accuracy, robustness, or security. Due to the fact that the context to be monitored might evolve over time, e.g., requirements or/and entities might change, some variables cannot be bounded at design time. Therefore, all possible values need to be checked, a task which is infeasible (Gupta and Schumann, 2004; Haydarlou et al., 2007; Goldsby et al., 2008).

The latter aspect is addressed by a couple of solution proposals for runtime V&V frameworks. The various approaches rely on the idea that some of the unbounded variables at design time can be instantiated at runtime. This spectrum includes the use of self-testable components, the adaption of test plans, depending on the instantiated configuration, the application of lightweight formal verification techniques that can deal with incomplete models, as well as the use of finite state machine models or combining design by contract with reflection mechanisms to extract the data to be verified during runtime. The techniques for online monitoring could make it easier to apply runtime V&V frameworks, as they might provide the information needed to instantiate some of the variables that are unbound at design time, but which can be restricted or defined at runtime (King et al., 2011; Mongiello et al., 2015; Sharifloo and Spoletini, 2013; Fredericks and Cheng, 2015a; Lee et al., 2018a).

Various solution proposals in the form of publications particularly address the sensing stage, dealing with continuous experimentation and feedback loops. In order to close the loops between the development team, the system, and the various stakeholders, some publications in our result set refer to solution proposals to adopt experiment-driven development practices, supporting the idea of the minimal product (Lenarduzzi and Taibi, 2016). However, with anticipatory systems, such as robots, cars with automated driving features, and smart contracts, conducting experiments may result in life-threatening situations or financial losses. To counteract this narrower scope for continuous experimentation (Bosch and Olsson, 2016c), such experiments could be augmented with simulations. First, this will help to extend the boundaries of experimenting with such systems and result in a better understanding of the system behavior, i.e., simulation-driven exploration of the behavior. Second, the simulations can help to detect misbehavior in the system at a rather early stage of system development (e.g., before the real hardware might be available).

In our result set, a couple of solution proposals address model-based and simulation-based testing. The discussed solutions often involve Petri Nets to capture complex behavior, but also use the occurrence of anomalies such as intelligent actioning to re-configure an autonomous system. To address the complexity of the environment/context of autonomous systems, a two-phase model has been proposed. First, the structural model captures the actors and their attributes, functions, and relations. Second, the behavioral model describes the possible states and transitions for these actors (Andrews et al., 2016; Abdelgawad et al., 2017; Eberhardinger et al., 2016b; Saglietti et al., 2015a; Saglietti and Meitner, 2016).

The importance of non-functional properties due to IoT and the need to validate and verify stochastic behavior at runtime has revealed solution proposals in the field of runtime quantitative verification. For anticipatory systems, the probabilistic models being used need to be updated based on observations of the behavior. The research community has developed verification libraries and probabilistic model checkers that rely on stochastic finite state transition models. Properties are expressed in probabilistic temporal logic. In addition to these novel developments, some publications in our result set suggest extending the conventional software engineering techniques in order to adapt them to RQV. For example, as anticipatory systems are able to anticipate future state, it is reasonable to use idle CPU time to pre-verify states that are likely to occur (Gerasimou et al., 2014; Calinescu et al., 2017b,a).

Research on testing agents and multi-agent systems has been widely carried out. Most of this research involves studies on agents that fit into the framework of monitoring, analyzing, planning, and execution. As this phases can be mapped onto the four key stages we used for classification, our seed set contains numerous articles on testing agent systems. Among the proposed solutions is an approach for the coverage-based testing of agents (covering code blocks) and to augment the rules of an agent such that the test efficiency can be improved. Another line of research involves simulation-based testing, whereby the agent's behavior and its environment are simulated (Gücan and Bernon, 2013; da Costa et al., 2010; Bai et al., 2011).

Finally, this discussion has been provided to increase the understanding of the V & V of anticipatory systems; it should not be interpreted as an attempt to evaluate the usefulness of the proposed techniques. The publications contained in the result set address important topics in testing anticipatory systems.

6. Related research

To our best knowledge, we are not aware of a secondary study on testing anticipatory systems. Here, we discuss secondary studies in areas that are related to the trends and results presented herein.

Testing context-ware applications use environmental information to provide better service to the actors or users. Such systems cover at least one of the four system type facets introduced herein. Matalonga et al. (2015) evaluate testing approaches for context-aware software and map the identified techniques to categories specified in ISO/IEC/IEEE 29119. Out of 1680 articles initially identified as relevant, they retrieved 110 for a detailed analysis. One of the goals of performing this systematic review was to map the identified techniques to the ISO/IEC/IEEE 29119 categories and to determine the degree to which (if at all) these techniques help in testing context-aware software. The authors conclude that most publications report on using well-known testing techniques to test context-aware applications. However, Matalonga et al. (2015) point out that they were not able to identify a test design technique that specifically targets context variables. The authors ask whether this is because these challenges have not been explored yet or because testing context-aware applications is no different from testing a (non context-aware) software system. According to Matalonga et al. (2015), further research is needed to answer this question.

Mattos et al. (2017a) present a quasi-systematic literature review performed to characterize testing methods for context-aware software systems. The authors identify evidence of methods that could be used to test context-aware applications and to gain insights into their effectiveness. First, Mattos et al. (2017a) conclude that a consensus is lacking, as almost half of the identified publications did not define the term context or context-aware. Second, the authors point out that context and test environment are two different concepts. To test such systems, Mattos et al. (2017a) argue that context is an intrinsic property of the SuT that should not be controlled. In contrast, the test environment can and should be defined at time of test design and needs to be implemented when dynamically executing the tests. Third, the authors conclude that a consensus is lacking regarding the meaning of software testing. The analyzed primary sources had different interpretations of the term software testing. According to the authors, the definition of software testing, according to ISO/IEC/IEEE 29119 international standard, allows almost every sort of quality assurance technique to be defined as software testing and thus is kept broad.

Agent systems are distributed systems consisting of agents that autonomously interact with each other in a complex environment; therefore, agents fit into the category of anticipatory systems as characterized in this article. Most recently, Bakar and Selamat (2018) conducted a systematic literature review of agent systems verification. They classified the proposed technologies according to three life cycle levels: design, development, and runtime. According to their study, design techniques such as model checking/formal methods, theorem proving, and mathematical/statistical have been proposed. For the development phase, the authors conclude that testing and debugging and simulation are the primary verification techniques. For runtime verification, Bakar and Selamat (2018) list fault management/detection, testing/debugging during execution, and runtime verification. Considering the number of publications, model checking/formal methods is followed by development testing/debugging and runtime verification. Among the most frequently checked properties are the system properties temporal correctness and liveness and correctness and safety. The authors provide a map of number of verification techniques with respect to the number of checked agent properties and detected faults.

Szvetits and Zdun (2016) present a systematic literature review performed to analyze objectives, techniques, kinds, and architectures for using models at runtime. In their work, the authors identify different objectives such as adaption, abstraction, consistency, conformance, error handling, monitoring, simulation, prediction, platform independence, and policy checking/enforcement. Regarding V&V, Szvetits and Zdun (2016) point out that runtime modes may be used for debugging, fault localization, tracing, self-healing, and test case generation and optimization. According to the authors, state machine models and workflow models are particularly suitable for checking against given execution traces due to their expressive nature in declaring data and execution flows (Szvetits and Zdun, 2016). Policy checking and enforcement can be carried out with runtime models to verify time constraints, safety properties, access control and security regulations, or other compliance rules.

7. Threats to validity

This study suffers from some general threats to validity. First, the study may be incomplete, as we have chosen the search strings based on our experience and performed the search in a single database. To counteract this risk, we used Scopus, which claims to be the largest database of abstracts and citations. Moreover, we have complemented our search strategy by performing forward and backward snowballing with Google Scholar. In addition, two of the three authors have a long history of experience in the testing and AI field, which counteracts the risk of choosing inadequate search strings. A preliminary study enabled us to refine and extend the used search strings. As we mentioned before, our decision to use neural networks over machine learning in the search string may reduce the number of publications and lead to a smaller result set. Similarly, not explicitly considering system testing in the search string may also impact the publications considered in this study.

Second, the study may have a general publication bias, as positive results are more likely to be published than negative results. Hence, our study may draft a potentially incomplete and overly positive view of the state of the art. Negative results are, if at all, mostly reported in terms of lessons learned. To counteract the second risk, we carefully examined trends observed regarding the contribution type in our preliminary and extended studies. Concerning our result set, the lessons learned have been reported regularly (see Fig. 2) for each of the system types.

Furthermore, like every survey, our study is exposed to threats regarding internal validity and external validity. We address these threats below.

Internal validity

First, our study encountered some systematic risks, such as the fact that it could be biased by personal ratings. Second, we chose the focus type to characterize a wide range of anticipatory systems. This incurs the risk of misclassification. To counteract the first aspect, all classification tasks were carried out by two authors, and we resolved the few conflicting assignments by implementing a voting process among all three authors. We counteracted the second risk by allowing multiple assignments regarding the system type and by using a dedicated guideline to assure the quality of the mapping process. Such guidelines contribute to objectifying the classification, as they provide specific criteria for evaluation by example.

External validity

The external validity is threatened by missing knowledge about the generalization of results. As we have focused on systems that exhibit anticipatory behavior, a risk of overlooking results is present that a broader analysis may have prevented. Regarding external validity and generalization, the results and analysis are limited to testing anticipatory systems. We addressed external validity due to two major countermeasures. First, we considered publications that contribute to verification and validation, as testing addresses both of these activities. This increased the number of publications in the result set, and we provide a view that is as general as possible regarding this emerging field of research. Second, to increase the external validity, further updates and/or replication studies are required to confirm the findings presented herein. For this reason, we provide the consolidated result set to make it possible for other researchers to replicate and complement this study.

8. Conclusion

Currently, we are witnessing a dramatic and broad technological and economic shift, as all of the technology that is required to transform industries through software is put in place. With anticipatory software steering our cars, smart agents deciding how to distribute energy across the power grid, and software-powered drones delivering goods, systems that exhibit anticipatory behavior are increasingly controlling devices that may harm humans or making decisions that are critical for business. In contrast, testing such systems has received little attention from either the artificial intelligence or the software engineering community. The mapping study described in this article was carried out to investigate three major research questions (RQs):

- RQ 1: What research and contribution types are found on the subject of testing anticipatory systems, and how many papers have been published for each type?
- RQ 2: What types of anticipatory systems have been addressed in the literature?
- RQ 3: What are the trends and open issues regarding the testing techniques/methods for anticipatory systems?

As a secondary study, this work also made the following contributions.

First, we identified a pool of relevant literature (result set) on the topics of testing anticipatory systems. We observe a trend regarding an increasing frequency of publications addressing testing of anticipatory systems. Recently, particular solution proposals in the form of methods/frameworks outperform this general trend. Only a few articles could be identified that address theory.

Second, we characterize the obtained result set by introducing three dimensions: the research type, the contribution type, and the system type. Regarding the latter, 73 (out of 206) articles in our result set describe investigations of SuT and cover all four key stages (sensing, inferring, predicting context, and intelligent actioning) of anticipatory systems. Our work also provides a systematic map for these types and presents a discussion about the observed quantitative and qualitative issues.

Third, we identify trends in testing anticipatory applications. Among these are:

- online monitoring
- automated test case generation
- test process and life cycle
- agent testing
- runtime verification and validation
- model-based and simulation-based testing
- continuous experimentation and feedback loops
- quantitative verification

For future research, we want to focus on online monitoring together with runtime verification and validation, and automated test case generation in the context of autonomous driving. Monitoring and checking the behavior of autonomous cars during operation is considerably important to prevent from the consequences of faults not detected during testing after development. For this purpose, we believe that formalized knowledge could allow us to detect critical situations. In addition, it is necessary to provide testing approaches that assures identifying critical scenarios and corner cases. Both activities are of uttermost importance for improving the quality of anticipatory systems.

CRediT authorship contribution statement

Bernhard Peischl: Conceptualization, Investigation, Data curation, Writing – original draft. **Oliver A. Tazl:** Investigation, Data curation, Writing – review & editing. **Franz Wotawa:** Validation, Writing – review & editing, Supervision, Funding acquisition.

Table 14
The result set obtained from the database search after applying all filter criteria.

Author (year)	Title
Abuseta and Swesi (2015)	Towards a framework for testing and simulating self adaptive systems
Akour et al. (2011)	Towards change propagating test models in autonomic and adaptive systems
Albers and Düser (2010)	A new process for configuring and applying complex validation environments using the example of vehicle-in-the-loop at the roller test bench
Arnold and Alexander (2013)	Testing autonomous robot control software using procedural content generation
Berger (2015)	Design considerations for a cyber–physical testing language with the example of autonomous driving
Berger (2013)	Improving scenario selection for simulations by run-time control-flow analysis
Berger (2012)	Accelerating Regression Testing for Scaled Self-Driving Cars with Lightweight Virtualization-A Case Study
Borck et al. (2017)	100 years of software - adapting cyber–physical systems to the changing world
Broderick (2005b)	Adaptive verification for an on-line learning neural-based flight control system
Chander et al. (1999)	Incremental and integrated evaluation of rule-based systems
Diebelis and Bicevskis (2010)	Test points in self-testing
Fredericks and Cheng (2015b)	Automated Generation of Adaptive Test Plans for Self-Adaptive Systems
Geddes (1991)	Verification and validation testing of the pilot's associate
Gulati et al. (2015)	Resolving ADAS imaging subsystem functional safety quagmire
Hänsel and Giese (2015)	A testing scheme for self-adaptive software systems with architectural runtime models
Hussein et al. (2012)	An Approach to Specifying and Validating Context-aware Adaptive Behaviours of Software Systems
Nehring and Liggesmeyer (2013)	Testing the Reconfiguration of Adaptive Systems
King et al. (2011)	Safe runtime validation of behavioral adaptations in autonomic software
Kondratenko et al. (2017)	Computerized system for remote level control with discrete self-testing
Laval et al. (2013)	A methodology for testing mobile autonomous robots
Lee et al. (2018b)	RINGA: Design and verification of finite state machine for self-adaptive software at runtime
Lee et al. (2017b)	Runtime verification method for self-adaptive software using reachability of transition system model
Li et al. (2018b)	Artificial intelligence test: a case study of intelligent vehicles
Markov and Frohlich (2015)	An ecosystem approach for testing self-organizing, adaptive systems
Mattos et al. (2017b)	More for less: Automated experimentation in software-intensive systems
Mauritz et al. (2016)	Assuring the safety of advanced driver assistance systems through a combination of simulation and runtime monitoring
Mei et al. (2012)	Preemptive regression test scheduling strategies: A new testing approach to thriving on the volatile service environments
Moran et al. (2017)	Towards Ex Vivo Testing of MapReduce Applications
Mutanu and Kotonya (2018)	A self-learning approach for validation of runtime adaptation in service-oriented systems
Nguyen et al. (2008c)	ECAT: A tool for automating test cases generation and execution in testing multi-agent systems
Nguyen et al. (2008d)	Ontology-based test generation for multi-agent systems
Olsson (2008)	Testing and verification of adaptive cruise control and collision warning with brake support by using HIL simulations
Pezzè and Wuttke (2009)	Automatic Generation of Runtime Failure Detectors from Property Templates
Püschel et al. (2014)	A Black Box Validation Strategy for Self-adaptive Systems
Robert (2017)	First insights into testing autonomous robot in virtual worlds
Saglietti et al. (2015b)	Reconfiguration testing for cooperating autonomous agents
Schneider and Trapp (2011)	A safety engineering framework for open adaptive systems
Schneider et al. (2016)	Gaining certainty about uncertainty: Testing cyber–physical systems in the presence of uncertainties at the application level
Taylor et al. (2003)	Verification and validation of neural networks: A sampling of research in progress
Wienke and Wrede (2017b)	Performance regression testing and run-time verification of components in robotics systems
Zhang et al. (2009)	Automated Testing for Intelligent Agent Systems
Nakajima (2018)	Quality Assurance of Machine Learning Software
Guo et al. (2018)	DLFuzz: Differential fuzzing testing of deep learning systems
Rao et al. (2019)	An approach for validating safety of perception software in autonomous driving systems
Kuutti et al. (2019)	Safe Deep Neural Network-Driven Autonomous Vehicles Using Software Safety Cages
Chechik et al. (2019)	Software assurance in an uncertain world
Sun et al. (2019b)	Structural test coverage criteria for deep neural networks
Sun et al. (2019a)	DeepConcolic: Testing and debugging deep neural networks
Ma et al. (2018)	DeepGauge: Multi-granularity testing criteria for deep learning systems
Damiani and Ardagna (2020)	Certified Machine-Learning Models
Wotawa (2018)	On the automation of testing a logic-based diagnosis system
Nishi et al. (2018)	A test architecture for machine learning product
Pei et al. (2019)	Deepxplore: Automated whitebox testing of deep learning systems
Elgharbawy et al. (2019c)	Ontology-based adaptive testing for automated driving functions using data mining techniques
Elgharbawy et al. (2019a)	Adaptive functional testing for autonomous trucks
Elgharbawy et al. (2019b)	A testing framework for predictive driving features with an electronic Horizon
Bozic et al. (2019)	Chatbot Testing Using AI Planning
Guichard et al. (2019)	Assessing the Robustness of Conversational Agents using Paraphrases
Byun et al. (2019)	Input Prioritization for Testing Neural Networks
Marijan et al. (2019)	Challenges of Testing Machine Learning Based Systems
Zhu et al. (2019)	Datamorphic Testing: A Method for Testing Intelligent Applications

Declaration of competing interest

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

Acknowledgments

Financial support from the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, Austria and the Christian Doppler Research Association, Austria is gratefully acknowledged.

Appendix

See Table 14.

References

Andreessen, M., 2011. Why software is eating the world. Wall Street J. https://www.wsj.com/.

Babaoglu, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A.P.A., van Steen, M. (Eds.), 2005. Self-Star Properties in Complex Information Systems, Conceptual and Practical Foundations [the Book Is a Result from a Workshop At Bertinoro, Italy, Summer 2004]. In: volume 3460 of Lecture Notes in Computer Science, Springe., http://dx.doi.org/10.1007/b136551.

Baresi, L., Ghezzi, C., 2010. The disappearing boundary between development-time and run-time. In: Roman, G., Sullivan, K.J. (Eds.), Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, At the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November (2010) 7-11. ACM, pp. 17–22. http://dx.doi.org/10.1145/1882362.1882367, http://doi.acm.org/10.1145/1882362.1882367.

Beizer, B., 1990. Software Testing Techniques. Van Nostrand Reinhold.

Broderick, R.L., 2005a. Adaptive verification for an on-line learning neural-based flight control system. In: 24th Digital Avionics Systems Conference, Vol. 1. (6.C.2–61–10), http://dx.doi.org/10.1109/DASC.2005.1563392.

Budgen, D., Turner, M., Brereton, P., Kitchenham, B., 2008. Using mapping studies in software engineering. In: Proceedings of PPIG. Lancaster University, pp. 195–204.

Butz, M.V., Sigaud, O., Géard, P., 2003. Anticipatory Behavior in Adaptive Learning Systems, Foundations, Theories, and Systems. In: volume 2684 of Lecture Notes in Computer Science, Springer, http://dx.doi.org/10.1007/b11711.

de Sousa Santos, I., de Castro Andrade, R.M., Rocha, L.S., Matalonga, S., de Oliveira, K.M., Travassos, G.H., 2017. Test case design for context-aware applications: Are we there yet? Inf. Softw. Technol. 88, 1–16. http://dx.doi.org/10.1016/j.infsof.2017.03.008.

Eagle, N., Pentland, A., 2006. Reality mining: Sensing complex social systems. Pers. Ubiquitous Comput. 10, 255–268. http://dx.doi.org/10.1007/s00779-005-0046-3.

Feyzmahdavian, H.R., Besselink, B., Johansson, M., 2018. Stability analysis of monotone systems via max-separable Lyapunov functions. IEEE Trans. Automat. Contr. 63, 643–656. http://dx.doi.org/10.1109/TAC.2017.2727282.

Fredericks, E.M., Cheng, B.H.C., 2015a. Automated generation of adaptive test plans for self-adaptive systems. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, Piscataway, NJ, USA, pp. 157–168.

Gücan, O., Bernon, C., 2013. A generic testing framework for agent-based simulation models. 7. J. Simulation 7 (3), 183–201. http://dx.doi.org/10.1057/jos.2012.26.

Haghighatkhah, A., Banijamali, A., Pakanen, O., Oivo, M., Kuvaja, P., 2017. Automotive software engineering: A systematic mapping study. J. Syst. Softw. 128, 25–55. http://dx.doi.org/10.1016/j.jss.2017.03.005.

Hänsel, J., Vogel, T., Giese, H., 2015. A testing scheme for self-adaptive software systems with architectural runtime models. In: 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 134–139. http://dx.doi.org/10.1109/SASOW.2015.27.

Harrison, R., Mernik, M., Henriques, P., Cruz, D.da., Menzies, T., Rodriguez, D., 2013. 2Nd international workshop on realizing artificial intelligence synergies in software engineering (raise 2013). In: 2013 35th International Conference on Software Engineering. ICSE, pp. 1543–1544. http://dx.doi.org/10.1109/ICSE.2013.6606778.

Harrison, R., Rodriguez, D., Henriques, P., 2012. Welcome to the first international workshop on realizing artificial intelligence synergies in software engineering (raise 2012). In: 2012 First International Workshop on Realizing AI Synergies in Software Engineering. RAISE, pp. iii–iv. http://dx.doi.org/10.1109/RAISE.2012.6227960.

Kinsner, W., 2009. Challenges in the design of adoptive, intelligent and cognitive systems. IJSSCI 1, 16–35. http://dx.doi.org/10.4018/jssci.2009070102.

Kitchenham, A., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University.

Kohavi, R., Deng, A., Frasca, B., Longbotham, R., Walker, T., Xu, Y., 2012. Trustworthy online controlled experiments: Five puzzling outcomes explained. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, pp. 786–794. http://dx.doi.org/10.1145/2339530.2339653, http://doi.acm.org/10.1145/2339530.2339653.

Kuhrmann, M., Diebold, P., Müch, J., 2016. Software process improvement: A systematic mapping study on the state of the art. PeerJ Comput. Sci. 2, e62. http://dx.doi.org/10.7717/peerj-cs.62.

Kwiatkowska, M.Z., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July (2011) 14-20. Proceedings. Springer, pp. 585–591. http://dx.doi.org/10.1007/978-3-642-22110-1_47.

Lee, E., Kim, Y.G., Seo, Y.D., Seol, K., Baik, D.K., 2017a. Runtime verification method for self-adaptive software using reachability of transition system model. In: Proceedings of the Symposium on Applied Computing. ACM, New York, NY, USA, pp. 65–68. http://dx.doi.org/10.1145/3019612.3019851, http://doi.acm.org/10.1145/3019612.3019851.

Lee, E., Kim, Y.G., Seo, Y.D., Seol, K., Baik, D.K., 2018a. Ringa: Design and verification of finite state machine for self-adaptive software at runtime. Inf. Softw. Technol. 93, 200–222. http://dx.doi.org/10.1016/j.infsof.2017.09.008, http://www.sciencedirect.com/science/article/pii/S0950584917304792.

Lenarduzzi, V., Taibi, D., 2016. Mvp explained: A systematic mapping study on the definitions of minimal viable product. In: 42th Euromicro Conference on Software Engineering and Advanced Applications. SEAA 2016, Limassol, Cyprus, August 31 - Sept. 2, 2016, IEEE Computer Society, pp. 112–119. http://dx.doi.org/10.1109/SEAA.2016.56.

Li, L., Lin, Y.L., Zheng, N.N., Wang, F.Y., Liu, Y., Cao, D., Wang, K., Huang, W.L., 2018a. Artificial intelligence test: A case study of intelligent vehicles. Artif. Intell. Rev. http://dx.doi.org/10.1007/s10462-018-9631-5.

Lim, Y.J., Jee, E., Shin, D., Bae, D.H., 2015. Efficient testing of self-adaptive behaviors in collective adaptive systems. In: 2015 IEEE 39th Annual Computer Software and Applications Conference. pp. 216–221. http://dx.doi.org/10.1109/COMPSAC.2015.131.

Liu, Y., Cukic, B., Fuller, E., Yerramalla, S., Gururajan, S., 2006. Monitoring techniques for an online neuro-adaptive controller. J. Syst. Softw. 79, 1527–1540. http://dx.doi.org/10.1016/j.jss.2006.03.048, software Cybernetics, http://www.sciencedirect.com/science/article/pii/S016412120600121X.

Liu, Y., Cukic, B., Gururajan, S., 2007. Validating neural network-based online adaptive systems: A case study. Softw. Qual. J. 15, 309–326. http://dx.doi.org/10.1007/s11219-007-9017-4.

Low, C.K., Chen, T.Y., Rónnquist, R., 1999. Automated test case generation for bdi agents. Auton. Agents Multi-Agent Syst. 2, 311–332. http://dx.doi.org/10.1023/A:1010011219782.

Markov, G., Fröhlich, J., 2015. An ecosystem approach for testing self-organizing, adaptive systems. In: 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 120–121. http://dx.doi.org/10.1109/SASOW.2015.24.

Matalonga, S., Rodrigues, F., Travassos, G.H., 2015. Matching context aware software testing design techniques to iso/iec/ieee 29119. In: Rout, T., O'Connor, R.V., Dorling, A. (Eds.), Software Process Improvement and Capability Determination. Springer International Publishing, Cham, pp. 33–44.

Mattos, D.I., Bosch, J., Olsson, H.H., 2017a. More for less: Automated experimentation in software-intensive systems. In: Felderer, M., Fernández, D.M., Turhan, B., Kalinowski, M., Sarro, F., Winkler, D. (Eds.), Product-Focused Software Process Improvement - 18th International Conference. PROFES 2017, Innsbruck, Austria, November 29 - December 1, 2017, Proceedings, Springer, pp. 146–161. http://dx.doi.org/10.1007/978-3-319-69926-4_12.

Minku, L., Miransky, A., Turhan, B., 2016. Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE@ICSE 2016, Austin, Texas, USA, May (2016) 14-22. ACM, http://dx.doi.org/10.1145/2896995, http://doi.acm.org/10.1145/2896995.

Mongiello, M., Pelliccione, P., Sciancalepore, M., 2015. Ac-contract: Run-time verification of context-aware applications. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, Piscataway, NJ, USA, pp. 24–34.

Mourão, E., Kalinowski, M., Murta, L., Mendes, E., Wohlin, C., 2017. Investigating the use of a hybrid search strategy for systematic reviews. In: Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE Press, Piscataway, NJ, USA, pp. 193–198. http://dx.doi.org/10.1109/ESEM.2017.30.

Munoz, F., Baudry, B., 2009. Artificial table testing dynamically adaptive systems. CoRR abs/0903.0914. http://arxiv.org/abs/0903.0914.

Nadin, M., 2010. Anticipatory computing. from a high-level theory to hybrid computing implementations. Int. J. Appl. Res. Inf. Technol. Comput. (IJARITAC) 1, 1–27.

Nadin, M., 2014. Quantifying anticipatory characteristics. the anticipationscope™ and the anticipatoryprofile™. In: Iantovics, B., Kountchev, R. (Eds.), Advanced Intelligent Computational Technologies and Decision Support Systems. In: volume 486 of Studies in Computational Intelligence, Springer.

Nguyen, C.D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M., 2012. Evolutionary testing of autonomous software agents. Auton. Agents Multi-Agent Syst. 25, 260–283. http://dx.doi.org/10.1007/s10458-011-9175-4.

Nguyen, C.D., Perini, A., Tonella, P., 2008a. Ecat: A tool for automating test cases generation and execution in testing multi-agent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC.. pp. 1669–1670, http://dl.acm.org/citation.cfm?id=1402744.1402757.

Nguyen, C.D., Perini, A., Tonella, P., 2008b. Ontology-based test generation for multiagent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Vol. 3. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC., pp. 1315–1320, http://dl.acm.org/citation.cfm?id=1402821.1402860.

Olsson, H.H., Bosch, J., 2014. From opinions to data-driven software r & d: A multi-case study on how to close the 'open loop' problem.

Padgham, L., Zhang, Z., Thangarajah, J., Miller, T., 2013. Model-based test oracle generation for automated unit testing of agent systems. IEEE Trans. Softw. Eng. 39, 1230–1244. http://dx.doi.org/10.1109/TSE.2013.10.

Patel, K., Hierons, R.M., 2017. A mapping study on testing non-testable systems. Softw. Qual. J. 10.1007/s11219-017-9392-4.

Peischl, B., Tazl, O.A., Wotawa, F., 2017. Testing of artificial intelligence applications: A state of the art survey. In: Gams, M., Zupančič, J. (Eds.), Proceedings of the 20th International Multiconference Information Society - IS 2017, AS-IT-IC Workshop, Volume E. pp. 23–26.

Pejovic, V., Musolesi, M., 2015. Anticipatory mobile computing: A survey of the state of the art and research challenges. ACM Comput. Surv. 47, 47:1–47:29. http://dx.doi.org/10.1145/2693843, http://doi.acm.org/10.1145/2693843.

Péez-Sánchez, B., Fontenla-Romero, O., Guijarro-Berdiñas, B., 2014. Adaptive neural topology based on vapnik-chervonenkis dimension. In: Duval, B., van den Herik, H.J., Loiseau, S., Filipe, J. (Eds.), Agents and Artificial Intelligence - 6th International Conference. ICAART 2014, Angers, France, March (2014) 6-8, Revised Selected Papers, Springer, pp. 194–210. http://dx.doi.org/10.1007/978-3-319-25210-0_12.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18. http://dx.doi.org/10.1016/j.infsof.2015.03.007, http://www.sciencedirect.com/science/article/pii/S0950584915000646.

Qin, Y., Xu, C., Yu, P., Lu, J., 2016. Sit: Sampling-based interactive testing for self-adaptive apps. J. Syst. Softw. 120, 70–88. http://dx.doi.org/10.1016/j.jss.2016.07.002, http://www.sciencedirect.com/science/article/pii/S0164121216301029.

Reichstaller, A., Knapp, A., 2017. Transferring context-dependent test inputs. In: 2017 IEEE International Conference on Software Quality, Reliability and Security. QRS, pp. 65–72. http://dx.doi.org/10.1109/QRS.2017.16.

Rodríguez P. Haghighatkhah, A. and Lwakatare, L.E. and Teppola, S. and Suomalainen, T. and Eskeli, J. and Karvonen, T. and Kuvaja, P. and Verner, J.M. and Oivo, M., title=Continuous deployment of software intensive products and services: A systematic mapping study, J. Syst. Softw. 123, 263–291. http://dx.doi.org/10.1016/j.jss.2015.12.015.

Rosen, R., Rosen, J., Kineman, J., Nadin, M., 2012. Anticipatory Systems: Philosophical, Mathematical, and Methodological Foundations. In: IFSR International Series in Systems Science and Systems Engineering, Springer, New York.

Saglietti, F., Meitner, M., 2016. Model-driven structural and statistical testing of robot cooperation and reconfiguration. In: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, ACM, New York, NY, USA. pp. 17–23. http://dx.doi.org/10.1145/3022099.3022102, http://doi.acm.org/10.1145/3022099.3022102.

Saglietti, F., Winzinger, S., Lill, R., 2015a. Reconfiguration testing for cooperating autonomous agents. In: Koornneef, F., van Gulijk, C. (Eds.), Computer Safety, Reliability, and Security. Springer International Publishing, Cham, pp. 144–155.

Schumann, J., Gupta, P., Jacklin, S., 2005. Toward verification and validation of adaptive aircraft controllers. In: 2005 IEEE Aerospace Conference. pp. 1–6. http://dx.doi.org/10.1109/AERO.2005.1559606.

Schumann, J., Liu, Y., 2007. Tools and methods for the verification and validation of adaptive aircraft control systems. In: 2007 IEEE Aerospace Conference. pp. 1–8. http://dx.doi.org/10.1109/AERO.2007.352766.

Sharifloo, A.M., Spoletini, P., 2013. Lover: Light-weight formal verification of adaptive systems at run time. In: Păăreanu, C.S., Salaün, G. (Eds.), Formal Aspects of Component Software. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 170–187.

Szvetits, M., Zdun, U., 2016. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. Softw. Syst. Model. 15, 31–69. http://dx.doi.org/10.1007/s10270-013-0394-9.

Tamura, G., Villegas, N.M., Müler, H.A., Sousa, J.P., Becker, B., Karsai, G., Mankovski, S., Pezzè, M., Schäfer, W., Tahvildari, L., Wong, K., 2010. Towards practical runtime verification and validation of self-adaptive software systems. In: de Lemos, R., Giese, H., Müler, H.A., Shaw, M. (Eds.), Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October (2010) 24-29 Revised Selected and Invited Papers. Springer, pp. 108–132. http://dx.doi.org/10.1007/978-3-642-35813-5_5.

Taylor, Brian J., Marjorie A. Darrah, C.D.M., 2003. Verification and validation of neural networks: a sampling of research in progress. In: Proc.SPIE 5103, Vol. 9. p. 5103. http://dx.doi.org/10.1117/12.487527.

Turhan, B., Bener, A., Harrison, R., Miransky, A., Mericli, C., Minku, L., 2015. 4Th international workshop on realizing ai synergies in software engineering. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. raise 2015, pp. 991–992. http://dx.doi.org/10.1109/ICSE.2015.320.

Turhan, B., Bener, A.B., Meriçi, Ç., Miranskyy, A.V., Minku, L.L. (Eds.), 2014. 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2014, Hyderabad, India, June 3, 2014. ACM, http://dl.acm.org/citation.cfm?id=2593801.

Visnevski, N.A., Castillo-Effen, M., 2010. Evolutionary computing for mission-based test and evaluation of unmanned autonomous systems. In: 2010 IEEE Aerospace Conference. pp. 1–10. http://dx.doi.org/10.1109/AERO.2010.5446782.

Wang, S., Zhu, H., 2012. Catest: A test automation framework for multi-agent systems. In: 2012 IEEE 36th Annual Computer Software and Applications Conference. pp. 148–157. http://dx.doi.org/10.1109/COMPSAC.2012.24.

Wienke, J., Wrede, S., 2017a. Performance regression testing and run-time verification of components in robotics systems. Adv. Robot. 31, 1177–1192. http://dx.doi.org/10.1080/01691864.2017.1395360.

Wieringa, R., Maiden, N.A.M., Mead, N.R., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11, 102–107. http://dx.doi.org/10.1007/s00766-005-0021-6.

Winikoff, M., 2017. Bdi agent testability revisited. Auton. Agents Multi-Agent Syst. 31, 1094–1132. http://dx.doi.org/10.1007/s10458-016-9356-2.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, New York, NY, USA, pp. 38:1–38:10. http://dx.doi.org/10.1145/2601248.2601268, http://doi.acm.org/10.1145/2601248.2601268.

Wotawa, F., Peischl, B., Klük, F., Nica, M., 2018. Quality assurance methodologies for automated driving. Elektrotech. Informationstech. 135, 322–327. http://dx.doi.org/10.1007/s00502-018-0630-7.

Yang, W., Xu, C., Pan, M., Cao, C., Ma, X., Lu, J., 2018. Efficient validation of self-adaptive applications by counterexample probability maximization. J. Syst. Softw. 138, 82–99. http://dx.doi.org/10.1016/j.jss.2017.12.009, http://www.sciencedirect.com/science/article/pii/S0164121217303023.

Zhang, J., Cheng, B.H.C., 2006. Using temporal logic to specify adaptive program semantics. J. Syst. Softw. 79, 1361–1369. http://dx.doi.org/10.1016/j.jss.2006.02.062.

Zhang, Z., Thangarajah, J., Padgham, L., 2008. Automated unit testing intelligent agents in pdt. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC., pp. 1673–1674.

Zhang, Z., Thangarajah, J., Padgham, L., 2011. Automated testing for intelligent agent systems. In: Gleizes, M.P., Gomez-Sanz, J.J. (Eds.), Agent-Oriented Software Engineering X. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 66–79.

Zofka, M.R., Klemm, S., Kuhnt, F., Schamm, T., Zöllner, J.M., 2016. Testing and validating high level components for automated driving: simulation framework for traffic scenarios. In: 2016 IEEE Intelligent Vehicles Symposium. IV, 144–150, http://dx.doi.org/10.1109/IVS.2016.7535378.

Zofka, M.R., Kohlhaas, R., Schamm, T., Zöllner, J.M., 2014. Semivirtual simulations for the evaluation of vision-based adas. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings. pp. 121–126. http://dx.doi.org/10.1109/IVS.2014.6856593.

Result set & primary studies

Abuseta, Y., Swesi, K., 2015. Towards a framework for testing and simulating self adaptive systems. In: 2015 6th IEEE International Conference on Software Engineering and Service Science. ICSESS, pp. 70–76. http://dx.doi.org/10.1109/ICSESS.2015.7339008.

Akour, M., Jaidev, A., King, T.M., 2011. Towards change propagating test models in autonomic and adaptive systems. In: 18th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. ECBS 2011, Las Vegas, NV, USA, 27-29 April, 2011, IEEE Computer Society, pp. 89–96. http://dx.doi.org/10.1109/ECBS.2011.23.

Albers, A., Düser, T., 2010. A new process for configuration and application of complex validation environments using the example of vehicle-in-the-loop at the roller test bench. In: ASME International Mechanical Engineering Congress and Exposition, Proceedings. IMECE, pp. 807–816. http://dx.doi.org/10.1115/IMECE2010-39959.

Arnold, J., Alexander, R., 2013. Testing autonomous robot control software using procedural content generation. In: Bitsch, F., Guiochet, J., Kaâiche, M. (Eds.), Computer Safety, Reliability, and Security - 32nd International Conference, SAFECOMP 2013, Toulouse, France, September (2013) 24-27. Proceedings. Springer, pp. 33–44. http://dx.doi.org/10.1007/978-3-642-40793-2_4.

Berger, C., 2012. Design considerations for a cyber–physical testing language on the example of autonomous driving. In: Proceedings of the 2012 Workshop on Domain-Specific Modeling. ACM, New York, NY, USA, pp. 49–54. http://dx.doi.org/10.1145/2420918.2420932, http://doi.acm.org/10.1145/2420918.2420932.

Berger, C., 2013. Improving scenario selection for simulations by run-time control-flow analysis. In: Bruzzone, A.G., Kropf, P., Riley, L.A., Davoudpour, M., Solis, A.O. (Eds.), 2013 Summer Simulation Multiconference, SummerSim '13, Toronto, Canada - July (2013) 07-10. Society for Computer Simulation International / ACM DL, p. 25.

Berger, C., 2015. Accelerating regression testing for scaled self-driving cars with lightweight virtualization - a case study. In: Bures, T., Weyns, D., Klein, M., Haber, R.E. (Eds.), 1st IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems. SEsCPS 2015, Florence, Italy, May 17, 2015, IEEE Computer Society, pp. 2–7. http://dx.doi.org/10.1109/SEsCPS.2015.9.

Borck, H., Kline, P., Shackleton, H., Gohde, J., Johnston, S., Alexander, P., Carpenter, T., 2017. 100 Years of software - adapting cyber–physical systems to the changing world. In: Romanovsky, A.B., Troubitsyna, E. (Eds.), Software Engineering for Resilient Systems - 9th International Workshop. SERENE 2017, Geneva, Switzerland, September (2017) 4-5, Proceedings, Springer, pp. 133–148. http://dx.doi.org/10.1007/978-3-319-65948-0_9.

Bozic, J., Tazl, O.A., Wotawa, F., 2019. Chatbot testing using ai planning. In: 2019 IEEE International Conference on Artificial Intelligence Testing. pp. 37–44. http://dx.doi.org/10.1109/AITest.2019.00-10.

Broderick, R.L., 2005b. Adaptive verification for an on-line learning neural-based flight control system. In: 24th Digital Avionics Systems Conference. pp. 6.C.2–61–10 1. http://dx.doi.org/10.1109/DASC.2005.1563392.

Byun, T., Sharma, V., Vijayakumar, A., Rayadurgam, S., Cofer, D., 2019. Input prioritization for testing neural networks. In: 2019 IEEE International Conference on Artificial Intelligence Testing. pp. 63–70. http://dx.doi.org/10.1109/AITest.2019.000-6.

Chander, P.G., Shinghal, R., Radhakrishnan, T., 1999. Incremental and integrated evaluation of rule-based systems. In: Imam, Ibrahim F., Kodratoff, Yves, El-Dessouki, Ayman, Ali, Moonis (Eds.), Multiple Approaches to Intelligent Systems, 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-99, Cairo, Egypt, May 31 - June 3, 1999, Proceedings. Springer, pp. 276–285. http://dx.doi.org/10.1007/3-540-48765-4_31.

Chechik, M., Salay, R., Viger, T., Kokaly, S., Rahimi, M., 2019. Software assurance in an uncertain world. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11424 LNCS, pp. 3–21. http://dx.doi.org/10.1007/978-3-030-16722-6_1.

Damiani, E., Ardagna, C., 2020. Certified Machine-Learning Models. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 12011 LNCS, pp. 3–15. http://dx.doi.org/10.1007/978-3-030-38919-2_1, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079096357&doi=10.1007%2f978-3-030-38919-2_1&partnerID=40&md5=1194df7ca18ffb215d038558ec58503c.

Diebelis, E., Bicevskis, J., 2010. Test points in self-testing, in: Barzdins, J., Kirikova, M. (Eds.), Databases and Information Systems VI - Selected Papers from the Ninth International Baltic Conference, DB & IS 2010, July (2010) 5-7, Riga, Latvia, IOS Press. pp. 309–321. http://dx.doi.org/10.3233/978-1-60750-688-1-309, https://doi.org/10.3233/978-1-60750-688-1-309.

Elgharbawy, M., Scherhaufer, I., Oberhollenzer, K., Frey, M., Gauterin, F., 2019a. Adaptive functional testing for autonomous trucks. Int. J. Transport. Sci. Technol. 8, 202–218. http://dx.doi.org/10.1016/j.ijtst.2018.11.003.

Elgharbawy, M., Schwarzhaupt, A., Arenskrieger, R., Elsayed, H., Frey, M., Gauterin, F., 2019b. A testing framework for predictive driving features with an electronic horizon. Transp. Res. F 61, 291–304. http://dx.doi.org/10.1016/j.trf.2017.08.002.

Elgharbawy, M., Schwarzhaupt, A., Frey, M., Gauterin, F., 2019c. Ontology-based adaptive testing for automated driving functions using data mining techniques. Transp. Res. F 66, 234–251. http://dx.doi.org/10.1016/j.trf.2019.07.021.

Fredericks, E.M., Cheng, B.H.C., 2015b. Automated generation of adaptive test plans for self-adaptive systems. In: Inverardi, P., Schmerl, B.R. (Eds.), 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS 2015, Florence, Italy, May (2015) 18-19, IEEE Computer Society, pp. 157–167. http://dx.doi.org/10.1109/SEAMS.2015.15.

Geddes, N.D., 1991. Verification and validation testing of the pilot's associate. In: IEEE/AIAA 10th Digital Avionics Systems Conference. pp. 426–431. http://dx.doi.org/10.1109/DASC.1991.177204.

Guichard, J., Ruane, E., Smith, R., Bean, D., Ventresque, A., 2019. Assessing the robustness of conversational agents using paraphrases. In: 2019 IEEE International Conference on Artificial Intelligence Testing. pp. 55–62. http://dx.doi.org/10.1109/AITest.2019.000-7.

Gulati, R., Easwaran, V., Karandikar, P., Mody, M., Shankar, P., 2015. Resolving adas imaging subsystem functional safety quagmire. In: 2015 IEEE International Conference on Consumer Electronics. ICCE, pp. 291–294. http://dx.doi.org/10.1109/ICCE.2015.7066419.

Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J., 2018. Dlfuzz: Differential fuzzing testing of deep learning systems. pp. 739–743. http://dx.doi.org/10.1145/3236024.3264835, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058304120&doi=10.1145%2f3236024.3264835&partnerID=40&md5=ce34bac0d915f65df8a0af97678ad7c6.

Hänsel, T., Giese, H., 2015. A testing scheme for self-adaptive software systems with architectural runtime models. In: 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 134–139. http://dx.doi.org/10.1109/SASOW.2015.27.

Hussein, M., Han, J., Colman, A., Yu, J., 2012. An approach to specifying and validating context-aware adaptive behaviours of software systems. In: 9th IEEE International Conference and Workshops on the Engineering of Autonomic & Autonomous Systems, Novi Sad, Serbia.

King, T.M., Allen, A.A., Cruz, R., Clarke, P.J., 2011. Safe runtime validation of behavioral adaptations in autonomic software. In: Calero, J.M.A., Yang, L.T., Mámol, F.G., García-Villalba, L.J., Li, X.A. and Wang, Y. (Eds.), Autonomic and Trusted Computing - 8th International Conference, ATC 2011, Banff, Canada, September (2011) 2-4. Proceedings. Springer, pp. 31–46. http://dx.doi.org/10.1007/978-3-642-23496-5_3.

Kondratenko, Y.P., Kozlov, O.V., Topalov, A.M., Gerasin, O.S., 2017. Computerized system for remote level control with discrete self-testing. In: CEUR Workshop Proceedings. pp. 608–619.

Kuutti, S., Bowden, R., Joshi, H., Temple, R.de., Fallah, S., 2019. Safe Deep Neural Network-Driven Autonomous Vehicles using Software Safety Cages. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11872 LNCS, pp. 150–160. http://dx.doi.org/10.1007/978-3-030-33617-2_17, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85076959441&doi=10.1007%2f978-3-030-33617-2_17&partnerID=40&md5=62946248501bdf62beb0ee975306b953.

Laval, J., Fabresse, L., Bouraqadi, N., 2013. A methodology for testing mobile autonomous robots. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1842–1847. http://dx.doi.org/10.1109/IROS.2013.6696599.

Lee, E., Kim, Y., Seo, Y., Seol, K., Baik, D., 2017b. Runtime verification method for self-adaptive software using reachability of transition system model. In: Seffah, A., Penzenstadler, B., Alves, C., Peng, X. (Eds.), Proceedings of the Symposium on Applied Computing. SAC 2017, Marrakech, Morocco, April (2017) 3-7, ACM, pp. 65–68. http://dx.doi.org/10.1145/3019612.3019851, http://doi.acm.org/10.1145/3019612.3019851.

Lee, E., Kim, Y.G., Seo, Y.D., Seol, K., Baik, D.K., 2018b. Ringa: Design and verification of finite state machine for self-adaptive software at runtime. Inf. Softw. Technol. 93, 200–222. http://dx.doi.org/10.1016/j.infsof.2017.09.008, http://www.sciencedirect.com/science/article/pii/S0950584917304792.

Li, L., Lin, Y., Zheng, N., Wang, F., Liu, Y., Cao, D., Wang, K., Huang, W., 2018b. Artificial intelligence test: A case study of intelligent vehicles. Artif. Intell. Rev. 1–25. http://dx.doi.org/10.1007/s1046.

Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y., 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. pp. 120–131. http://dx.doi.org/10.1145/3238147.3238202, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056490436&doi=10.1145%2f3238147.3238202&partnerID=40&md5=fb27d226f969e544a090a66b0dd16408.

Marijan, D., Gotlieb, A., Kumar Ahuja, M., 2019. Challenges of testing machine learning based systems. In: 2019 IEEE International Conference on Artificial Intelligence Testing. pp. 101–102. http://dx.doi.org/10.1109/AITest.2019.00010.

Markov, G., Frohlich, J., 2015. An ecosystem approach for testing self-organizing, adaptive systems. In: 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASO Workshops 2015, Cambridge, MA, USA, September (2015) 21-25. IEEE Computer Society, pp. 120–121. http://dx.doi.org/10.1109/SASOW.2015.24.

Mattos, D.I., Bosch, J., Olsson, H.H., 2017b. More for less: Automated experimentation in software-intensive systems. In: Felderer, M., Fernádez, D.M., Turhan, B., Kalinowski, M., Sarro, F., Winkler, D. (Eds.), Product-Focused Software Process Improvement - 18th International Conference. PROFES 2017, Innsbruck, Austria, November 29 - December 1, 2017, Proceedings, Springer, pp. 146–161. http://dx.doi.org/10.1007/978-3-319-69926-4_12.

Mauritz, M., Howar, F., Rausch, A., 2016. Assuring the safety of advanced driver assistance systems through a combination of simulation and runtime monitoring. In: Margaria, T., Steffen, B. (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October (2016) 10-14, Proceedings, Part II. pp. 672–687. http://dx.doi.org/10.1007/978-3-319-47169-3_52.

Mei, L., Zhai, K., Jiang, B., Chan, W.K., Tse, T.H., 2012. Preemptive regression test scheduling strategies: A new testing approach to thriving on the volatile service environments.

Moran, J., Bertolino, A., Riva, C.de.la., Tuya, J., 2017. Towards ex vivo testing of mapreduce applications. In: 2017 IEEE International Conference on Software Quality, Reliability and Security. QRS, pp. 73–80. http://dx.doi.org/10.1109/QRS.2017.17.

Mutanu, L., Kotonya, G., 2018. A self-learning approach for validation of runtime adaptation in service-oriented systems. Serv. Orient. Comput. Appl. 12, 11–24. http://dx.doi.org/10.1007/s11761-017-0222-0.

Nakajima, S., 2018. Quality assurance of machine learning software. pp. 143–144. http://dx.doi.org/10.1109/GCCE.2018.8574766, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060315582&doi=10.1109%2fGCCE.2018.8574766&partnerID=40&md5=c235e7dc920981a2d526e0ad8a77ca39.

Nehring, K., Liggesmeyer, P., 2013. Testing the reconfiguration of adaptive systems. In: The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications. pp. 14–19.

Nguyen, C.D., Perini, A., Tonella, P., 2008c. Ecat: A tool for automating test cases generation and execution in testing multi-agent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 1669–1670.

Nguyen, C.D., Perini, A., Tonella, P., 2008d. Ontology-based test generation for multiagent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Vol. 3. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC., pp. 1315–1320.

Nishi, Y., Masuda, S., Ogawa, H., Uetsuki, K., 2018. A test architecture for machine learning product. pp. 273–278. http://dx.doi.org/10.1109/ICSTW.2018.00060.

Olsson, P., 2008. Testing and verification of adaptive cruise control and collision warning with brake support by using hil simulations. In: SAE World Congress & Exhibition. SAE International, http://dx.doi.org/10.4271/2008-01-0728.

Pei, K., Cao, Y., Yang, J., Jana, S., 2019. Deepxplore: Automated whitebox testing of deep learning systems. Commun. ACM 62, 137–145. http://dx.doi.org/10.1145/3361566.

Pezzè, M., Wuttke, J., 2009. Automatic generation of runtime failure detectors from property templates. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (Eds.), Software Engineering for Self-Adaptive Systems [Outcome of a Dagstuhl Seminar]. Springer, pp. 223–240. http://dx.doi.org/10.1007/978-3-642-02161-9_12.

Püschel, G., Piechnick, C., Götz, S., Seidl, C., Richly, S., Aßmann, U., 2014. A black box validation strategy for self-adaptive systems. In: Proceedings of the Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications. Citeseer., pp. 111–116.

Rao, D., Pathrose, P., Huening, F., Sid, J., 2019. An Approach for Validating Safety of Perception Software in Autonomous Driving Systems. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11842 LNCS, pp. 303–316. http://dx.doi.org/10.1007/978-3-030-32872-6_20.

Robert, C., 2017. First insights into testing autonomous robot in virtual worlds. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, pp. 112–115. http://dx.doi.org/10.1109/ISSREW.2017.59.

Saglietti, F., Winzinger, S., Lill, R., 2015b. Reconfiguration testing for cooperating autonomous agents. In: Koornneef, F., van Gulijk, C. (Eds.), Computer Safety, Reliability, and Security - SAFECOMP 2015 Workshops, ASSURE, DECSoS, ISSE, ReSA4CI, and SASSUR, Delft, the Netherlands, September 22, 2015, Proceedings. Springer, pp. 144–155. http://dx.doi.org/10.1007/978-3-319-24249-1_13.

Schneider, D., Trapp, M., 2011. A safety engineering framework for open adaptive systems. In: 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems. SASO 2011, Ann Arbor, MI, USA, October (2011) 3-7, IEEE Computer Society, pp. 89–98. http://dx.doi.org/10.1109/SASO.2011.20.

Schneider, M.A., Wendland, M., Bornemann, L., 2016. Gaining certainty about uncertainty - testing cyber–physical systems in the presence of uncertainties at the application level. In: Großmann, J., Felderer, M., Seehusen, F. (Eds.), Risk Assessment and Risk-Driven Quality Assurance - 4th International Workshop, RISK 2016, Held in Conjunction with ICTSS 2016, Graz, Austria, October 18, 2016, Revised Selected Papers. pp. 129–142. http://dx.doi.org/10.1007/978-3-319-57858-3_10.

Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R., 2019a. Deepconcolic: Testing and debugging deep neural networks. pp. 111–114. http://dx.doi.org/10.1109/ICSE-Companion.2019.00051, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85071847364&doi=10.1109%2fICSE-Companion.2019.00051&partnerID=40&md5=795124d9e697e63e7a9a98f06e109b4d.

Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R., 2019b. Structural test coverage criteria for deep neural networks. ACM Trans. Embedded Comput. Syst. 18, http://dx.doi.org/10.1145/3358233.

Taylor, Brian J., A., Marjorie, Darrah, C.D.M., 2003. Verification and validation of neural networks: a sampling of research in progress. http://dx.doi.org/10.1117/12.487527,DOI:10.1117/12.487527.

Wienke, J., Wrede, S., 2017b. Performance regression testing and run-time verification of components in robotics systems. Adv. Robot. 31, 1177–1192. http://dx.doi.org/10.1080/01691864.2017.1395360.

Wotawa, F., 2018. On the automation of testing a logic-based diagnosis system. pp. 370–373. http://dx.doi.org/10.1109/ICSTW.2018.00075, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85050994878&doi=10.1109%2fICSTW.2018.00075&partnerID=40&md5=16f2bbca4cfb41d69a21d4272c23ac94.

Zhang, Z., Thangarajah, J., Padgham, L., 2009. Automated testing for intelligent agent systems. In: Gleizes, M.P., Góez-Sanz, J.J. (Eds.), Agent-Oriented Software Engineering X - 10th International Workshop, AOSE 2009, Budapest, Hungary, May (2009) 11-12, Revised Selected Papers. Springer, pp. 66–79. http://dx.doi.org/10.1007/978-3-642-19208-1_5.

Zhu, H., Liu, D., Bayley, I., Harrison, R., Cuzzolin, F., 2019. Datamorphic testing: A method for testing intelligent applications. In: 2019 IEEE International Conference on Artificial Intelligence Testing. pp. 149–156. http://dx.doi.org/10.1109/AITest.2019.00018.

Abdelgawad, M., McLeod, S., Andrews, A., Xiao, J., 2017. Model-based testing of a real-time adaptive motion planning system. Adv. Robot. 31, 1159–1176. http://dx.doi.org/10.1080/01691864.2017.1396921.

Andrews, A., Abdelgawad, M., Gario, A., 2016. World model for testing autonomous systems using petri nets. In: 2016 IEEE 17th International Symposium on High Assurance Systems Engineering. HASE, pp. 65–69. http://dx.doi.org/10.1109/HASE.2016.11.

Bach, J., Otten, S., Sax, E., 2016. Model based scenario specification for development and test of automated driving functions. In: 2016 IEEE Intelligent Vehicles Symposium. IV, pp. 1149–1155. http://dx.doi.org/10.1109/IVS.2016.7535534.

Bai, X., Chen, B., Ma, B., Gong, Y., 2011. Design of intelligent agents for collaborative testing of service-based systems. In: Proceedings of the 6th International Workshop on Automation of Software Test. ACM, pp. 22–28. http://dx.doi.org/10.1145/1982595.1982601.

Bakar, N.A., Selamat, A., 2018. Agent systems verification : Systematic literature review and mapping. Appl. Intell. 48, 1251–1274. http://dx.doi.org/10.1007/s10489-017-1112-z.

Bosch, J., 2012. Building products as innovation experiment systems. In: Cusumano, Michael A., Iyer, Bala, Venkatraman, N. (Eds.), Software Business. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 27–39.

Bosch, J., Eklund, U., 2012. Eternal embedded software: Towards innovation experiment systems. In: Margaria, Tiziana, Steffen, Bernhard (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October (2012) 15-18, Proceedings, Part I. Springer, pp. 19–31. http://dx.doi.org/10.1007/978-3-642-34026-0_3.

Bosch, J., Olsson, H.H., 2016. Data-driven continuous evolution of smart systems. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. ACM, pp. 28–34. http://dx.doi.org/10.1145/2897053.2897066.

Bosse, T., Lam, D.N., Barber, K.S., 2006. Automated analysis and verification of agent behavior. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. ACM, pp. 1317–1319. http://dx.doi.org/10.1145/1160633.1160876.

Broderick, R.L., 2004. Statistical and adaptive approach for verification of a neural-based flight control system. In: The 23rd Digital Avionics Systems Conference (IEEE Cat. No. 04CH37576). pp. 6.E.1–61–10 2. http://dx.doi.org/10.1109/DASC.2004.1390736.

Calinescu, R., Gerasimou, S., Johnson, K., Paterson, C., 2017a. Using runtime quantitative verification to provide assurance evidence for self-adaptive software. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (Eds.), Software Engineering for Self-Adaptive Systems III. Assurances. Springer International Publishing, pp. 223–248.

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

Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T., 2017b. Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2017.2738640.

Coelho, R., Kulesza, U., von Staa, A., Lucena, C., 2006. Unit testing in multi-agent systems using mock agents and aspects. In: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems. ACM, pp. 83–90. http://dx.doi.org/10.1145/1138063.1138079.

da Costa, A.D., Nunes, C., da Silva, V.T., Fonseca, B., de Lucena, C.J.P., 2010. Jaaf+t: A framework to implement self-adaptive agents that apply self-test. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, pp. 928–935. http://dx.doi.org/10.1145/1774088.1774280.

Eberhardinger, B., Habermaier, A., Hoffmann, A., Poeppel, A., Reif, W., 2016a. Toward integrated analysis amp;amp; testing of component-based, adaptive robot systems. In: 2016 IEEE International Conference on Software Quality, Reliability and Security Companion. QRS-C, pp. 301–302. http://dx.doi.org/10.1109/QRS-C.2016.45.

Eberhardinger, B., Habermaier, A., Seebach, H., Reif, W., 2016b. Back-to-back testing of self-organization mechanisms. In: Wotawa, F., Nica, M., Kushik, N. (Eds.), Testing Software and Systems. Springer International Publishing, pp. 18–35.

Gerasimou, S., Calinescu, R., Banks, A., 2014. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In: Engels, G., Bencomo, N. (Eds.), 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. ACM, pp. 115–124.

Goldsby, H.J., Cheng, B.H.C., Zhang, J., 2008. Amoeba-rt: Run-time verification of adaptive software. In: Giese, H. (Ed.), Models in Software Engineering. Springer Berlin Heidelberg, pp. 212–224.

Gruyer, D., Choi, S., Boussard, C., d'Andréa Novel, B., 2014. From virtual to reality, how to prototype, test and evaluate new adas: Application to automatic car parking. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings. pp. 261–267. http://dx.doi.org/10.1109/IVS.2014.6856525.

Gupta, P., Schumann, J., 2004. A tool for verification and validation of neural network based adaptive controllers for high assurance systems. In: Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings. pp. 277–278. http://dx.doi.org/10.1109/HASE.2004.1281757.

Haydarlou, A.R., Oey, M.A., Overeinder, B.J., Brazier, F.M.T., 2007. Use case driven approach to self-monitoring in autonomic systems. In: Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on. p. 50. http://dx.doi.org/10.1109/CONIELECOMP.2007.113.

Hernández-Orallo, J., 2017. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. Artif. Intell. Rev. 397–447. http://dx.doi.org/10.1007/s10462-016-9505-7.

Jahan, S., Marshall, A., Gamble, R., 2017. Embedding verification concerns in self-adaptive system code. In: 2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems. pp. 121–130. http://dx.doi.org/10.1109/SASO.2017.21.

Dr. Bernhard Peischl received a Ph.D in computer science (2004) from the Graz University of Technology, Austria. He has more than 15 years of experience in software development and verification and validation of software-enabled systems. He is currently innovation manager tasked with sourcing, setting up and managing national and European R&D programs within the AVL PTE business unit.

Bernhard has co-authored over 90 scientific articles on peer-reviewed workshops, conferences, scientific journals, and books. Since 2007 he is a lecturer at the Graz University of Technology covering topics such as software technology, artificial intelligence, and value-based software engineering.

Oliver A. Tazl received an M.Sc. from Graz University of Technology in 2017. He works as research and teaching assistant at the Institute for Software Technology at the Graz University of Technology. His research interests include model-based diagnosis, recommender systems, and verification and validation for AI-based systems. Oliver Tazl has been working in several research projects including AS-IT-IC where he worked on smart chatbot technology. He is author of 4 peer reviewed papers published at scientific workshops and conferences.

Franz Wotawa received an M.Sc. and a Ph.D. from Vienna University of Technologyin 1994 and 1996 respectively. He is a professor of software engineering at Graz University of Technology and head of the Institute for Software Technology and the Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber–Physical Systems. His research interests include model-based and qualitative reasoning, theorem proving, mobile robots, verification and validation, and software testing and debugging. During his career Franz Wotawa has written more than 390 peer-reviewed papers for journals, books, conferences, and workshops, and supervised 90 Master and 36 Ph.D. students.