# A Systematic Literature Mapping of Artificial Intelligence Planning in Software Testing

Luis F. de Lima, Leticia M. Peres, André R. A. Grégio and Fabiano Silva
*Department of Informatics, Federal University of Paraná, Av. Cel. Francisco H. dos Santos, 100, Curitiba, PR, Brazil*

Keywords: Software Testing, Artificial Intelligence Planning, AI Planning, Planning Technique.

Abstract: Software testing is one of the most expensive software development processes. So, techniques to automate this process are fundamental to reduce software cost and development time. Artificial intelligence (AI) planning technique has been applied to automate part of the software testing process. We present in this paper a systematic literature mapping (SLM), using Petersen et al. (2015) approach of methods, techniques and tools regarding AI planning in software testing. Using the mapping, we identify 16 papers containing methods, techniques, frameworks and tools proposals, besides a survey. We identify testing techniques, testing phases, artifacts, AI planning techniques, AI planning tools, support tools, and generated plans in these selected papers. By mapping data analyses we identify a deficiency in the use of white-box and error-based testing techniques, besides the recent use of AI planning in security testing.

## 1 INTRODUCTION

The main objective of the software testing process is to run a program in a controlled environment to reveal faults (Myers, 1979). This process consumes the most effort during software development (Pressman, 2016). In this way, it is essential to plan the software testing, allowing us to estimate how much work, time and resources will be needed. Besides, the growing demand for quality systems has motivated the search for new techniques, methods and testing tools.

Testing activity involves producing and running the system with test cases, which are indicated by testing criteria. Testing criteria are defined from testing techniques, such as functional, structural and error-based (Maldonado et al., 2007), occurring with the system functionalities knowledge, with the system internal logic knowledge, and through the most common system errors, respectively. These testing techniques are usually used in the context of unit, integration, system and acceptance testing phases (Pressman, 2016).

Several artificial intelligence (AI) techniques have been associated with the software testing techniques and phases. Srivastava and Kim (2009) use genetic algorithms in software testing. Malz et al. (2012) present a concept for an automated prioritization of test cases using software agents and fuzzy logic. Jiang et al. (2018) apply fuzzy logic in software test metrics to increase accuracy in software testing. Zhu and Jiao (2019) apply genetic algorithms to automate testing data generation.

Another AI technique used in software testing is AI planning. This technique consists in the automatic elaboration of a sequence of actions to reach an objective (Russell and Norvig, 2016). We can associate the sequential execution of actions of an AI planning problem with a test case execution, allowing the modeling of software testing that has the goal of finding unexpected system behavior.

The present study presents a systematic literature mapping (SLM) of methods, techniques and tools regarding the AI planning technique applied in software testing. We opted to perform an SLM as a method of scientific investigation that brings relevant studies of a research field. We based the SLM search method on Petersen et al. (2015) approach.

The main goal of this work is to provide a quantitative research area overview, identifying the frequency of publications over time, testing techniques, testing phases, used artifacts, AI planning techniques, AI planning tools, support tools, and generated artifacts. Another goal is to identify a weakness in the state-of-the-art related to AI planning in software testing.

We organized this paper as follows: Section 2 provides software testing concepts and AI planning definitions. Section 3 presents a systematic literature mapping methodology. Section 4 presents and discusses the mapping results. Section 5 presents related works. Section 6 presents the final considerations and discusses future work.

F. de Lima, L., Peres, L., Grégio, A. and Silva, F.
A Systematic Literature Mapping of Artificial Intelligence Planning in Software Testing.
DOI: 10.5220/0009829501520159
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 152-159
ISBN: 978-989-758-443-5

## 2 BACKGROUND

### 2.1 Software Testing

The software development process is divided into phases. A phase is a period in which activities with specific objectives are realized (Wazlawick, 2013). Software testing is one of the software development phases, performed with the application of testing criteria that are defined by different testing techniques.

The most common software testing techniques are black-box testing, white-box testing, and error-based testing (Maldonado et al., 2007). Black-box testing, also known as functional testing, permits software testing looking at its functionalities, disregarding its implementation. White-box testing, or structural testing, uses the software implementation and its internal structural knowledge for its testing. Error-based testing uses the most frequent errors in software development to perform its verification and validation.

Testing activity is usually divided into the following phases: unit testing, integration testing, system testing, and acceptance testing. In unit testing, each unit is a procedure or function in a procedural programming language or a class in an object-oriented programming language, and the programmer tests each unit separately from one another (Wazlawick, 2013).

Integration testing occurs when the units have already been tested separately and will be integrated into a new system version. System testing occurs with the system totally integrated, to verify possible errors according to the system specification. Integration testing and system testing are made by the testing team. Acceptance testing is performed by system end users, with the purpose of performing system validation regarding user requirements (Wazlawick, 2013).

### 2.2 Artificial Intelligence Planning

Artificial intelligence (AI) planning is defined as the automatic elaboration of an action sequence to achieve the aim of a problem (Russell and Norvig, 2016). A planning problem is described as a quadruple $(P, I, G, A)$ where $P$ is a set of predicates, $I$ is the initial state of the problem, $G$ is the goal state of the problem, and $A$ is a set of actions.

Predicates $p \in P$ represents a first-order logic formula and are used to define all actions $a \in A$. All states of the problem are specified by predicates that are true in these states. Actions are described in terms of preconditions and effects (functions $pre(a)$ and $post(a)$, respectively). If $pre(a)$ of an action $a$ is true in a state $S$ then $a$ is executed. After the action $a$ execution, the transition to another state $S'$ occurs, i.e., $S \xrightarrow{a} S'$. After the transition, $post(a)$ effects are automatically considered valid in state $S'$.

Plan possibly solves an AI planning problem. A plan is a set of $n$ actions whose sequential execution takes from the initial state to the goal state of the problem, i.e., $I \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 ... \xrightarrow{a_n} G$. Different plans can be generated according to the AI planning algorithm used to solve the problem. Plan is an artifact resulting from an AI planning tool execution.

Planning problems are usually represented by formal languages, e.g., STRIPS (Fikes and Nilsson, 1971), ADL (Pednault, 1989), and PDDL (McDermott et al., 1998). AI planning tools, also known as planners, use AI planning problems in formal languages as inputs to generate plans. Some planners are ABSTRIPS (Sacerdoti, 1974), which uses a STRIPS input, UCPOP (Penberthy et al., 1992), for ADL inputs, and Metric-FF (Hoffmann, 2003), for PDDL inputs.

## 3 METHODOLOGY OF SYSTEMATIC MAPPING

The main objective of a systematic literature mapping (SLM) isto summarize a field of interest through classification, concerned with structuring the research area (Petersen et al., 2008). We chose the Petersen et al. (2015) approach to conduct our SLM because these authors performed a SLM in the software engineering area. We composed our SLM methodology by five steps for definitions of objective, research questions, search strategy, studies selection and data extraction, described in the following subsections.

### 3.1 Step 1: Objective

The first step of the SLM is to identify its objective. This SLM aims to review and analyze the state-of-the-art of AI planning theories, implementations, methods, techniques, and tools in software testing.

### 3.2 Step 2: Research Questions

The following SLM step is to define the research questions (RQs). The RQs determine what should be answered at the end of the mapping. First, we define the following main RQ.

- **RQ**: How AI planning is used in software testing?

In addition, another eight RQs were defined to reach the SLM objective.

- **RQ1:** What is the proposal of AI planning use in software testing?

  If tool:
  - **RQ1.1:** What is the proposed tool?
  - **RQ1.2:** Is the tool's code open?
- **RQ2:** What is the testing phase?
- **RQ3:** What is the testing technique used?
- **RQ4:** What is the AI planning technique used?
- **RQ5:** What is the artifact used by AI planning?
- **RQ6:** What is the planner used?
- **RQ7:** What are the support tools used?
- **RQ8:** What does the generated plan represent?

## 3.3 Step 3: Search Strategy

The next methodology step is the search strategy definition. In this step, the search bases are chosen and the search strings are elaborated. We chose ACM[1], IEEE Xplore[2], Scopus[3] and Springer[4] databases. The following search string was used to search: "("Software Engineering" AND ("Software Test" OR "Software Testing") AND ("AI Planning" OR "Artificial Intelligence Planning"))". In each database the search string had some syntactic changes according to the base pattern.

We selected the "computer science" area and "software engineering" discipline in the databases that offered these options. Restrictions about type, location, and year of publication were not applied. Searches were conducted at the Department of Informatics of the Federal University of Paraná, and were initialized in the middle of September 2018 and completed early October 2018, resulting in 149 papers. Table 1 shows the number of search results per database. These 149 selected papers were submitted to study selection. As a complement to these results, an additional manual search was performed in November 2019, which is described below.

Table 1: Number of studies per database.

| Database | Search results |
|---|---|
| ACM Digital | 4 |
| IEEE | 136 |
| Scopus | 2 |
| SpringerLink | 7 |

[1]Available in: https://dl.acm.org/

[2]Available in: http://ieeexplore.ieee.org

[3]Available in: https://www.scopus.com

[4]Available in: https://link.springer.com

## 3.4 Step 4: Study Selection

We realized the studies selection step was realized after the searches in databases. We divide this methodology step into two phases. In the first phase, the titles, keywords and abstracts of all selected papers in the databases searches are read. After that, papers are selected according to include criteria (IC). The second phase exclude papers selected in the first phase according to exclude criteria (EC).

We used one criteria to include papers:

**IC1:** papers using AI planning in software testing.

And four criteria to exclude papers:

**EC1:** papers that are not fully available;

**EC2:** papers that appear in more than[] one database;

**EC3:** papers that are not in English; and

**EC4:** books and grey literature.

Although the search string defined is specific to AI planning, we found some false positives. As the search string presents "test" , "testing" and "planning", some works related to "testing planning" were found in the most varied areas. So, the IC application was important to studies selection. The IC1 application included 15 out of 149 papers selected in the database search. The ECs application has not eliminated any paper. So, we selected 15 papers in this process.

In addition, we also conducted research in Google Scholar[5] search engine in November 2019 performing manual technique and 1 more paper was included, increasing the number of included papers to 16. Selected papers publication occurred between 1995 and 2019. Figure 1 shows the number of papers during the studies selection process.

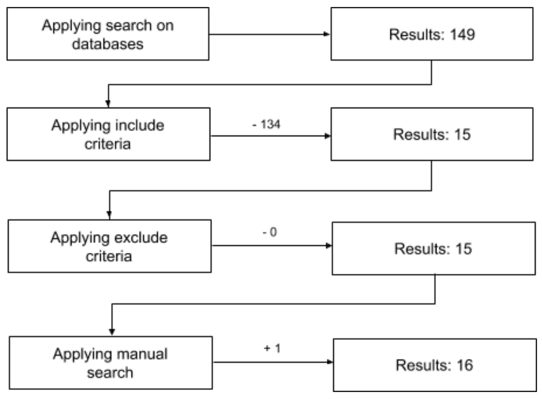
Applying search on databases
Results: 149
Applying include criteria
- 134
Results: 15
Applying exclude criteria
- 0
Results: 15
Applying manual search
+ 1
Results: 16

Figure 1: Number of included papers during the study selection step.

[5]Available in: https://scholar.google.com.br/

### 3.5 Step 5: Data Extraction

Data extraction is the last step of the SLM methodology. The 16 papers selected in Step 4 were submitted to full reading and data extraction. Based on the RQs defined, a data extraction template was elaborated, allowing posterior data classification and analysis. We associated an identifier (ID) to each paper, and identified the paper link (URL), paper title, paper authors, year of publication, authors department, authors country, information about publication congress/event, an overview of the paper proposal, and informations refers to the predefined RQs.

## 4 RESULTS AND DISCUSSION

This section presents and discusses the results obtained with data extraction. Data extraction results in the 15 papers selected in the SLM are presented by RQ. Manual searching results and discussions are presented in Subsection 4.8. Table 2 presents an overview of all selected papers.

### 4.1 AI Planning in Software Testing (RQ1)

In order to answer RQ1 we identified the following proposals of IA planning in software testing. Frameworks in (Feather and Smith, 2001), (Yen et al., 2002), (Razavi et al., 2014), and (Bozic and Wotawa, 2018); tools in (Memon et al., 2001), (Gupta et al., 2007), and (Li et al., 2009); models in (Mraz et al., 1995), (Howe et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999), (von Mayrhauser et al., 1999); and techniques in (Memon et al., 1999), (von Mayrhauser et al., 2000), and (Wotawa, 2016).

- **Tools and Codes (RQ1.1 and RQ1.2)**

Memon et al. (2001) propose Planning Assisted Tester for grapHical user interface Systems (PATHS), a graphical user interface test cases generation tool; Gupta et al. (2007) propose Means-Ends Analysis Graphplan (MEA-Graphplan), a test data generation tool that constructs the problem graph from the goal state to the initial state and then uses the standard Graphplan algorithm; Li et al. (2009) propose Multiple Fact Files - Interference Progression Planner (MF-IPP), a test case generation tool that divides the input file into smaller files for gaining performance.

The codes of these three tools identified were not made available by the paper's authors.

### 4.2 Testing Phases and Techniques (RQ2 and RQ3)

To answer RQ2 and RQ3 we analyzed the selected papers to identify phases and testing techniques. None of the papers makes explicit the testing phase where their proposals are made. Wotawa (2016) and Bozic and Wotawa (2018) refer to security testing, so it is implied that they run in the system testing phase. In von Mayrhauser et al. (2000) is used as an error-based testing technique and Razavi et al. (2014) use a structural testing technique. The other 13 papers use a functional (black-box) testing technique. So, 81% of the selected papers use the black-box testing technique in their proposals.

### 4.3 AI Planning Techniques (RQ4)

For answering RQ4 we analysed selected papers regarding AI planning techniques related to language and implementation that were used. In (Mraz et al., 1995), (Howe et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999), (von Mayrhauser et al., 1999), and (von Mayrhauser et al., 2000), the ADL language is used. Hierarchical Task Network (HTN) technique is used in (Memon et al., 2001). Planning Graph Analysis technique is used in (Gupta et al., 2007) and (Li et al., 2009). PDDL language is used in (Razavi et al., 2014) and (Bozic and Wotawa, 2018). STRIPS language is used in (Wotawa, 2016). In (Memon et al., 1999), (Feather and Smith, 2001), and (Yen et al., 2002) the AI planning techniques used are not specified.

### 4.4 Artifacts (RQ5)

Different artifacts were used as a base to model the planning problem in formal languages as planners input. For RQ5 we identified the following artifacts in the selected papers. StorageTek Robot Tape Library classes in (Mraz et al., 1995), (Howe et al., 1995), and (Howe et al., 1997); UML state and class diagrams in (Scheetz et al., 1999) and (von Mayrhauser et al., 1999); mutant operators in (von Mayrhauser et al., 2000); interface diagrams in (Feather and Smith, 2001); interface functions in (Memon et al., 1999) and (Memon et al., 2001); specification of system components in (Yen et al., 2002); finite state machine representing the system's states in (Gupta et al., 2007); user interface features in (Li et al., 2009); history of variables shared by competing Java programs and violation patterns in (Razavi et al., 2014); and XSS and SQL injection vulnerabilities descriptions in (Wotawa, 2016) and (Bozic and Wotawa, 2018).

Table 2: Selected papers overview.

| Paper | Overview |
|---|---|
| Mraz et al. (1995) | Model for test data generation for the StorageTek Robot Tape Library command language. |
| Howe et al. (1995) | Model for test data generation for the StorageTek Robot Tape Library. |
| Howe et al. (1997) | Model for test case generation for the StorageTek Robot Tape Library. |
| Scheetz et al. (1999) | Model for test case generation satisfying the UML-derived test objectives. |
| von Mayrhauser et al. (1999) | Model for test cases generation for system testing based on high level test objectives. |
| Memon et al. (1999) | Technique for automatic test cases generation for graphical user interfaces. |
| von Mayrhauser et al. (2000) | Technique for error recovery tests generation with concepts of mutation testing. |
| Feather and Smith (2001) | Framework for automatic generation of automated test oracle. |
| Memon et al. (2001) | Tool for automatic test cases generation for graphical user interfaces. |
| Yen et al. (2002) | Framework for assisting with the synthesis of code for assembling a system from existing components as well as automated testing of the system. |
| Gupta et al. (2007) | Tool for test data generation using the Graphplan algorithm. |
| Li et al. (2009) | Tool for test case generation able to handle multiple input files to avoid the combinatorial explosion. |
| Razavi et al. (2014) | Framework for predicting concurrent program runs containing data races, atomicity violations, or null-pointer references. |
| Wotawa (2016) | Technique for detecting vulnerabilities in systems. |
| Bozic and Wotawa (2018) | Framework for Web vulnerabilities modeling and security testing of Web applications. |
| Bozic and Wotawa (2019) | Survey describing 19 papers in the field of AI planning for software testing. |

## 4.5 Planners (RQ6)

For RQ6 we identified the following planners. Three different versions of UCPOP in (Mraz et al., 1995), (Howe et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999), (von Mayrhauser et al., 1999), and (von Mayrhauser et al., 2000); Interference Progression Planner (IPP) in (Memon et al., 1999); DS-1 in (Feather and Smith, 2001); Fast Forward (FF) in (Razavi et al., 2014); and JavaGP in (Bozic and Wotawa, 2018). Memon et al. (2001), Gupta et al. (2007), and Li et al. (2009) present their own tools proposals. Yen et al. (2002) and Wotawa (2016) did not specify the use of any planning tool.

## 4.6 Support Tools (RQ7)

For RQ7 we analyzed the selected papers to identify used support tools. Sleuth is a tool for automated application domain testing used by Mraz et al. (1995), Howe et al. (1995), and Howe et al. (1997) in the same planner experiments to compare the results obtained. WordPad graphical interface is used in (Memon et al., 1999) and (Memon et al., 2001) case studies. JPlan and IPP planners are used in (Gupta et al., 2007) and (Li et al., 2009) as the basis for the tools proposed by these authors. In order to gain performance in test case generation, Razavi et al. (2014) perform the inclusion of a planner in the Penelope testing tool's internal structure. Bozic and Wotawa (2018) use the HttpClient, jsoup and Crawler4j tools in their pentesting framework.

## 4.7 Plans (RQ8)

To answer RQ8 we analyzed the generated artifacts in the selected papers. These artifacts are plans resulting from planners execution like in (von Mayrhauser et al., 1999) and (Wotawa, 2016). In other works, each plan has different meanings according to the proposal of the paper. For example, in (Mraz et al., 1995), (Howe et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999), (Memon et al., 1999), (Memon et al., 2001), and (Li et al., 2009) the plans represent test cases. In (von Mayrhauser et al., 2000) the plan represents an error recovery test case. The remaining papers did not specify the resulting artifacts.

## 4.8 Manual Searching Results

In this section we discuss the paper (Bozic and Wotawa, 2019) that was identified with a manual search. These authors summarize without a SLM methodology 19 papers in the field of AI planning for software testing. The authors classified these papers into two groups: one related to functional testing containing 12 papers and other to non-functional testing (related to non-functional requirements such as performance, usability and security) containing 7 papers.

Among the 15 papers that we found in our SLM and the 19 papers reported by Bozic and Wotawa there is an intersection of 4 papers: (Mraz et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999), and (von Mayrhauser et al., 2000). The 15 papers listed by Bozic and Wotawa and not identified in our mapping are applied in areas such as network control, network systems, network protocols, Web services, Web applications, and industrial and automotive applications.

The papers listed by Bozic and Wotawa use some planning techniques that were not found in the papers of our SLM. For example, Anderson and Fickas (1989) use the adaptive planning technique, and Shmaryahu et al. (2018) use the contingent planning technique.

We identify 4 planners in the papers listed by Bozic and Wotawa that are not contained in our SLM: Bifrost, used in (Leitner and Bloem, 2005); Metric-FF, used in (Boddy et al., 2005), (Wotawa and Bozic, 2014), and (Bozic and Wotawa, 2018); SGplan used in (Obes et al., 2013); and Lama, used in (Schnelte and Güldali, 2010).

## 5 RELATED WORKS

In this section we present papers using AI planning techniques in other software development phases such as requirements engineering and design. The papers presented in this section were identified by conducting another SLM, with a variation of the methodology described in Section 3.

- **AI Planning in Requirements Engineering:** Amalio (2009) proposes a method for formal analysis of security requirements based on planning and uses the concept of suspicion to guide the search for threats and security vulnerabilities in requirements, using decreasoner planner. Liaskos et al. (2009) propose a framework to include the specification of optional user requirements and user preferences, using PPlan planner. Silva and Silva (2019) propose a method for requirements analysis based on a chain of algorithms and tools to allow features anticipated by knowledge engineering.
- **AI Planning in Design:** Honiden et al. (1994) propose a method to model the prototyping phases of systems in real-time as a planning problem, using Abstrips planner. Pérez and Crespo (2009) propose a method for computing refactoring sequences that can be directed to the system's structure problems correction that can negatively affect software quality factors. Soltani et al. (2011) and Soltani et al. (2012) propose frameworks to automatically select suitable features that satisfy the stakeholders' business concerns, resource limitations, functional and non-functional preferences, and constraints, using SHOP2 planner. Tunio et al. (2018) propose a method using PDDL language for task assignment in crowdsourcing software development.

## 6 CONCLUSIONS

Testing is a fundamental activity for software quality assurance. However, it is an activity involving a lot of time and resource effort during software development. Thus, there are constant proposals for new methods and techniques for its automation.

The objective of this paper was to investigate the use of AI planning as a technique for software testing automation. For this, we performed a systematic literature mapping (SLM), proving to be an effective way to identify, evaluate and interpret available relevant researches to a particular research question.

The main challenge of a SLM is to create a search string that does not limit the results. To ensure that the research covers as many works as possible with the string that we created, we did not include restrictions for research related to the type, location or period of publications.

Answering the main research question of the mapping, "how is AI planning used in software testing?", we found 15 papers published in the last three decades, proposing techniques, models, frameworks and tools. The remaining research questions of the SLM were answered by extracting data from the papers selected in the SLM. Testing techniques, testing phases, AI planning techniques, used artifacts in the planning problem definition, planners, support tools and the generated plans were identified.

Through additional manual research we found a survey listing papers in the field of AI planning for software testing. However, this paper presents the research survey without a SLM methodology application. This survey presents 19 papers, of which 15 differ from those found in the SLM presented in this work.

The SLM was conducted with a quantitative view, to identify weaknesses in the state-of-the-art of AI planning in software testing. With the extracted data analysis, we noticed many uses of the black-box testing technique. In this way, we identify a deficiency in the use of AI planning technique combined with the white-box testing and error-based testing. Another deficiency noted is about testing phases. Most of the selected papers did not specify or imply the testing phase in which their proposals are made.

The most recent papers found in the SLM showed a possible tendency of AI planning use in software security area. We observed this same fact in the survey found in the manual search. Most of these works use AI planning in penetration testing for vulnerabilities detection, also called pentesting. Another use is defining network protocols as AI planning problems to ensure the security of network systems and Web applications.

In order to obtain a view of AI planning in software engineering we conducted another SLM. Related works presented in this work were identified with this second SLM. In this other mapping, we found 8 papers using AI planning in requirements engineering and design phases. However, we identified a greater use of AI planning in the design phase (75 % of papers). With this second SLM results analysis, we noted a lack of tools proposals using AI planning to assist these software development phases.

Our objective with this work was to conduct a study on papers dealing with AI planning and software testing. With results analysis we noted a possible trend of AI planning use in security testing. In particular, we identified approaches to pentesting. We intend to take an approach to generate pentesting plans for Web applications, which will be our future work.

As another future works, we suggest approaches using white-box and error-based testing techniques associated with AI planning, and approaches at different testing phases to cover all tests performed throughout software development.

## ACKNOWLEDGEMENTS

We thank the financial support of C3SL (Department of Informatics/UFPR), PPGINF-CAPES/MEC and PET/MEC.

## REFERENCES

Amalio, N. (2009). Suspicion-Driven Formal Analysis of Security Requirements. *SECURWARE 2009*.

Anderson, J. S. and Fickas, S. (1989). A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. In *ACM SIGSOFT Software Engineering Notes*, number 3, pages 177–184. ACM.

Boddy, M. S., Gohde, J., Haigh, T., and Harp, S. A. (2005). Course of Action Generation for Cyber Security Using Classical Planning. In *ICAPS*, pages 12–21.

Bozic, J. and Wotawa, F. (2018). Planning-based Security Testing of Web Applications. In *Proceedings of the 13th International Workshop on Automation of Software Test*, pages 20–26. ACM.

Bozic, J. and Wotawa, F. (2019). Software Testing: According to Plan! In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 23–31. IEEE.

Feather, M. S. and Smith, B. (2001). Automatic Generation of Test Oracle from Pilot Studies to Application. *Automated Software Engineering*, 8(1):31–61.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial intelligence*, 2(3-4):189–208.

Gupta, M., Fu, J., Bastani, F. B., Khan, L. R., and Yen, I.-L. (2007). Rapid Goal-oriented Automated Software Testing Using MEA-graph Planning. *Software Quality Journal*, 15(3):241–263.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables. *Journal of artificial intelligence research*, 20:291–341.

Honiden, S., Nishimura, K., Uchihira, N., and Itoh, K. (1994). An Application of Artificial Intelligence to Object-oriented Performance Design for Real-time Systems. *IEEE transactions on software engineering*, (11):849–867.

Howe, A. E., von Mayrhauser, A., and Mraz, R. T. (1995). Test Sequences as Plans: An Experiment in Using an AI Planner to Generate System Tests. In *Knowledge-Based Software Engineering Conference, 1995. Proceedings., 10th*, pages 184–191. IEEE.

Howe, A. E., Von Mayrhauser, A., and Mraz, R. T. (1997). Test Case Generation as an AI Planning Problem. In *Knowledge-Based Software Engineering*, pages 77–106. Springer.

Jiang, D., Liang, Z., and Huang, D. (2018). A Software Test Metric Method Based on Fuzzy Logic. *DEStech Transactions on Computer Science and Engineering*, (iciti).

Leitner, A. and Bloem, R. (2005). Automatic Testing Through Planning. *Technische Universität Graz, Institute for Software Technology, Tech. Rep.*

Li, L., Wang, D., Shen, X., and Yang, M. (2009). A Method for Combinatorial Explosion Avoidance of AI Planner and the Application on Test Case Generation. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1–4. IEEE.

Liaskos, S., McIlraith, S. A., and Mylopoulos, J. (2009). Towards Augmenting Requirements Models with Preferences. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 565–569. IEEE Computer Society.

Maldonado, J. C., Barbosa, E. F., Vincenzi, A. M. R., Delamaro, M. E., Souza, S., and Jino, M. (2007). Introduction to Software Testing. *São Carlos*, page 23.

Malz, C., Jazdi, N., and Gohner, P. (2012). Prioritization of Test Cases Using Software Agents and Fuzzy Logic. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 483–486. IEEE.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL- The Planning Domain Definition Language.

Memon, A. M., Pollack, M. E., and Soffa, M. L. (1999). Using a Goal-driven Approach to Generate Test Cases for GUIs. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 257–266. IEEE.

Memon, A. M., Pollack, M. E., and Soffa, M. L. (2001). Hierarchical GUI Test Case Generation Using Automated Planning. *IEEE transactions on software engineering*, 27(2):144–155.

Mraz, R. T., Howe, A. E., von Mayrhauser, A., and Li, L. (1995). System Testing With an AI Planner. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pages 96–105. IEEE.

Myers, G. J. (1979). The Art of Software Testing. *ISBN: 0-471-04328-1*.

Obes, J. L., Sarraute, C., and Richarte, G. (2013). Attack Planning in the Real World. *arXiv preprint arXiv:1306.4044*.

Pednault, E. P. (1989). ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. *Kr*, 89:324–332.

Penberthy, J. S., Weld, D. S., et al. (1992). UCPOP: A Sound, Complete, Partial Order Planner for ADL. *Kr*, 92:103–114.

Pérez, J. and Crespo, Y. (2009). Perspectives on Automated Correction of Bad Smells. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 99–108. ACM.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Ease*, volume 8, pages 68–77.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update. *Information and Software Technology*, 64:1–18.

Pressman, R. (2016). *Software Engineering: A Practitioner's Approach, 8th ed.* McGraw Hill Brasil.

Razavi, N., Farzan, A., and McIlraith, S. A. (2014). Generating Effective Tests for Concurrent Programs via AI Automated Planning Techniques. *International Journal on Software Tools for Technology Transfer*, 16(1):49–65.

Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited.

Sacerdoti, E. D. (1974). Planning in a Hierarchy of Abstraction Spaces. *Artificial intelligence*, 5(2):115–135.

Scheetz, M., von Mayrhauser, A., and France, R. (1999). Generating Test Cases from an OO Model with an AI Planning System. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pages 250–259. IEEE.

Schnelte, M. and Güldali, B. (2010). Test Case Generation for Visual Contracts Using AI Planning. *INFORMATIK 2010. Service Science–Neue Perspektiven für die Informatik. Band 2*.

Shmaryahu, D., Shani, G., Hoffmann, J., and Steinmetz, M. (2018). Simulated Penetration Testing as Contingent Planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Silva, J. M. and Silva, J. R. (2019). A New Hierarchical Approach to Requirement Analysis of Problems in Automated Planning. *Engineering Applications of Artificial Intelligence*, 81:373–386.

Soltani, S., Asadi, M., Gašević, D., Hatala, M., and Bagheri, E. (2012). Automated Planning for Feature Model Configuration Based on Functional and Non-functional Requirements. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 56–65. ACM.

Soltani, S., Asadi, M., Hatala, M., Gasevic, D., and Bagheri, E. (2011). Automated Planning for Feature Model Configuration Based on Stakeholders Business Concerns. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 536–539. IEEE Computer Society.

Srivastava, P. R. and Kim, T.-h. (2009). Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications*, 3(4):87–96.

Tunio, M. Z., Luo, H., Wang, C., Zhao, F., Shao, W., and Pathan, Z. H. (2018). Crowdsourcing Software Development: Task Assignment Using PDDL Artificial Intelligence Planning. *Journal of Information Processing Systems*, 14(1).

von Mayrhauser, A., Scheetz, M., and Dahlman, E. (1999). Generating Goal-Oriented Test Cases. In *compsac*, page 110. IEEE.

von Mayrhauser, A., Scheetz, M., Dahlman, E., and Howe, A. E. (2000). Planner Based Error Recovery Testing. In *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*, pages 186–195. IEEE.

Wazlawick, R. (2013). *Software Engineering: Concepts and Practices*, volume 1. Elsevier Brasil.

Wotawa, F. (2016). On the Automation of Security Testing. In *Software Security and Assurance (ICSSA), 2016 International Conference on*, pages 11–16. IEEE.

Wotawa, F. and Bozic, J. (2014). Plan It! Automated Security Testing Based on Planning. In *IFIP International Conference on Testing Software and Systems*, pages 48–62. Springer.

Yen, I.-L., Bastani, F. B., Mohamed, F., Ma, H., and Linn, J. (2002). Application of AI Planning Techniques to Automated Code Synthesis and Testing. In *Tools with Artificial Intelligence, 2002.(ICTAI 2002). Proceedings. 14th IEEE International Conference on*, pages 131–137. IEEE.

Zhu, Z. and Jiao, L. (2019). Improving Search-Based Software Testing by Constraint-based Genetic Operators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1435–1442. ACM.