AI-T: SOFTWARE TESTING ONTOLOGY FOR AI-BASED SYSTEMS

J.I. Olszewska

School of Computing and Engineering, University of West Scotland, UK
joanna.olszewska@ieee.org


Abstract: Software testing is an expanding area which presents an increasing complexity. Indeed, on one hand, there is the development of technologies such as Software Testing as a Service (TaaS), and on the other hand, there is a growing number of Artificial Intelligence (AI)-based softwares. Hence, this work is about the development of an ontological framework for AI-softwares’ Testing (AI-T), which domain covers both software testing and explainable artificial intelligence; the goal being to produce an ontology which guides the testing of AI softwares, in an effective and interoperable way. For this purpose, AI-T ontology includes temporal interval logic modelling of the software testing process as well as ethical principle formalization and has been built using the Enterprise Ontology (EO) methodology. Our resulting AI-T ontology proposes both conceptual and implementation models and contains 708 terms and 706 axioms.

1 INTRODUCTION

Artificial Intelligence (AI) systems are nowadays pervasive in our Society, from face recognition (Raji et al., 2020) to cloud robotic systems (Pignaton de Freitas et al., 2020a). However, the black-box nature of many of these systems (Guidotti et al., 2018) leads to growing public concerns (Koene, 2017). Therefore, appropriate testing of such AI-based softwares is crucial to provide people and industry with dependable and ethical intelligent systems. Indeed, new-generation softwares are expected to be efficient, whilst transparent, safe, and secure (Olszewska, 2019b).

Software testing consists usually in a range of activities from manual testing to automated testing (Prasad, 2008), or Testing as a Service (TaaS) (Yu et al., 2010). They are applied throughout the software development life-cycle (SDLC) (Sommerville, 2015) and can be grouped into three processes, namely, the Organizational Test Process (OTP), the Test Management Process (TMP), and the Dynamic Test Process (DTP), which corresponding documentations report about (Alaqail and Ahmed, 2018).

Thus, OTP is utilized for the development and management of the organizational test specifications and includes three phases: to develop organizational test specification (OTP1); monitor and control the use of organizational test specifications (OTP2); and update organizational test specifications (OTP3). TMPs are used at the project level, and the three TMPs are defined as: test planning (TMP1); test monitoring and control (TMP2); and test completion (TMP3). On the other hand, the four DTPs comprise: test design & implementation (DTP1); test environment set-up & maintenance (DTP2); test execution (DTP3); and test incident reporting (DTP4). These DTPs are used to carry out dynamic testing within a particular level of testing (i.e. unit testing, integration testing, system testing, and acceptance testing) or type of testing (e.g. performance testing, security testing, usability testing) (IEEE, 2013a), (IEEE, 2013b), (IEEE, 2013c), (IEEE, 2015b), (IEEE, 2016).

In particular, ontologies are a convenient approach for capturing a conceptualization of the software testing domain (Tebes et al., 2019). Indeed, ontologies define terms, properties, relationships, and axioms explicitly, unambiguously, and in an interoperable way which could be understood by both humans and machines, since ontologies directly support automated reasoning about the domain knowledge representation (Chianese et al., 2009).

Hence, some ontologies have been developed for software testing (Ferreira de Souza et al., 2013a). These ontologies share common software testing knowledge, but differ among other, by their termino-
logical variations, since their terms are extracted from different body of knowledge, such as SoftWare Engineering Body of Knowledge (SWEBOK) (Freitas and Vieira, 2014), International Software Testing Qualifications Board (ISTQB) (Arnicans et al., 2013), and standards such as IEEE 610-1990 (Ferreira de Souza et al., 2013b), ISO/IEC 12207-1995 (Barbosa et al., 2006), ISO/IEC 9126-2001 (Cai et al., 2009), or IEEE 829-2008 (Vasanathapriyan et al., 2017).

Howbeit, many of these ontologies lack of axiomatization, e.g. (Guo et al., 2011), (Sapna and Mohanty, 2011), (Iliev et al., 2012), and some are rather mere taxonomies like (Cai et al., 2009) or (Anandaraj et al., 2011).

On the other hand, most of the software testing ontologies have a limited domain coverage, such as Web Service testing (Huo et al., 2003), Graphical User Interface (GUI) testing (Li et al., 2009), or Linux testing (Bezerra et al., 2018).

Moreover, no specific ontological methodologies have been applied to the development of ontologies such as Test Ontology Model (TOM) (Bai et al., 2008), Testing Maturing Model (TMM) ontology (Ryu et al., 2008), or Regression Test Execution (RTE) ontology (Campos et al., 2017).

These developed ontologies are useful, e.g. for determining dependability (Looker et al., 2005), proving interoperability (Yu et al., 2005), performing test reuse (Li and Zhang, 2012), test case generation (Crapo and Moitra, 2019), or test automation (Paydar and Kahani, 2010), but many of them do not have an implementation such as the Testing as a Service (TaaS) ontology (Yu et al., 2009).

Actually, very few software testing ontologies present both conceptual and implementation models, and these ontologies formalize a number of terms, e.g. 45 terms in the Reference Ontology on Software Testing (ROoST) (Ferreira de Souza et al., 2013b), 52 terms in OntoTest (Barbosa et al., 2006), 63 terms in the Software Testing Ontology for Web Service (STOWS) (Huo et al., 2003), and 194 terms in the Software Test Ontology (SwTO) (Bezerra et al., 2018).

Besides, ontologies have been designed for Artificial Intelligence (AI) domain (Bayat et al., 2016) and focus on intelligent agents (IA) (Valente et al., 2010), autonomous systems (AS) (Bermejo-Alonso et al., 2010), internet of things (IoT) (Ma et al., 2014), cyber-physical systems (CPS) (Dai et al., 2017), smart production systems (Terkaj and Urgo, 2014), or cloud robotic systems (CRS) (Pignaton de Freitas et al., 2020b).

However, no dedicated ontology has been developed for testing AI-based softwares.

In this paper, we propose a novel domain ontology for AI-softwares’ Testing (AI-T).

Its core knowledge includes the software testing domain as well as the ethical artificial intelligence domain.

AI-T ontology has been coded in Web Ontology Language Descriptive Logic (OWL DL), which is considered as the international standard for expressing ontologies and data on the Semantic Web (Guo et al., 2007), and using the Protege tool in conjunction with the FaCT++ reasoner (Tsarkov, 2014).

The resulting expert system provides an interoperable solution to test AI-based softwares.

Thus, the contributions of this paper is manyfold. Indeed, as far as we know, we propose the first ontology which purpose is to aid the testing of AI-based softwares. For that, we introduce the software ethical principle testing concepts along with the mathematical formalization of the software testing process using temporal-interval descriptive logic (DL). Besides, it is the first time Enterprise Ontology (EO) methodology is applied to develop a software testing ontology.

The paper is structured as follows. Section 2 presents the purpose and the building of our ontology for AI-based-software Testing (AI-T), while its evaluation and documentation are described in Section 3. Conclusions are drawn up in Section 4.

2 PROPOSED AI-T ONTOLOGY

To develop the AI-T ontology, we followed the ontological development life cycle (Gomez-Perez et al., 2004) based on the Enterprise Ontology (EO) Methodology (Dietz and Mulder, 2020), since EO is well suited for software engineering applications (van Kervel et al., 2012).

The adopted ontological development methodology consists of four main phases (Olszewska and Allison, 2018), which cover the whole development cycle, as follows:

1. identifications of the purpose of the ontology (Section 2.1);
2. ontology building which consists of three parts: the capture to identify the domain concepts and their relations; the coding to represent the ontology in a formal language; and the integration to share ontology knowledge (Section 2.2);
3. evaluation of the ontology to check that the developed ontology meets the scope of the project (Section 3.1);
4. documentation of the ontology (Section 3.2).
Table 1: Artificial-Intelligence-based software\(^*\) Testing (AI-T) Body of Knowledge (BoK) summary.

<table>
<thead>
<tr>
<th>Sample Concepts</th>
<th>Terminology Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Testing Strategy</td>
<td>SWEBOK, IEEE 29119-1-2013</td>
</tr>
<tr>
<td>Software Testing Process</td>
<td></td>
</tr>
<tr>
<td>Organizational Test Process</td>
<td>IEEE 29119-2-2013</td>
</tr>
<tr>
<td>Test Management Process</td>
<td></td>
</tr>
<tr>
<td>Dynamic Test Process</td>
<td></td>
</tr>
<tr>
<td>Software Testing Documentation</td>
<td>IEEE 29119-3-2013</td>
</tr>
<tr>
<td>Test Plan</td>
<td></td>
</tr>
<tr>
<td>Software Testing Level</td>
<td>ISTQB</td>
</tr>
<tr>
<td>Unit Testing</td>
<td></td>
</tr>
<tr>
<td>Integration Testing</td>
<td></td>
</tr>
<tr>
<td>System Testing</td>
<td></td>
</tr>
<tr>
<td>Acceptance Testing</td>
<td></td>
</tr>
<tr>
<td>Software Testing Technique</td>
<td>IEEE 29119-4-2015, IEEE 24765-2017</td>
</tr>
<tr>
<td>Dynamic Testing</td>
<td></td>
</tr>
<tr>
<td>Specification-Based Technique</td>
<td></td>
</tr>
<tr>
<td>Structure-Based Technique</td>
<td></td>
</tr>
<tr>
<td>Static Testing</td>
<td></td>
</tr>
<tr>
<td>Software Testing Type</td>
<td></td>
</tr>
<tr>
<td>Software Quality Testing</td>
<td>IEEE 29119-4-2015, (Sommerville, 2015)</td>
</tr>
<tr>
<td>Software Ethical Principle Testing</td>
<td>IEEE EADv2-2019, VDEv1-2020, IEEE 70xx</td>
</tr>
<tr>
<td>Application-Based Testing (e.g. AI Software)</td>
<td>IEEE 1872-2015, (Leonard et al., 2017), (Olszewska, 2019a)</td>
</tr>
<tr>
<td>Software Comparison Testing</td>
<td>(Sommerville, 2015)</td>
</tr>
<tr>
<td>Software Testing Measure &amp; Metric</td>
<td>(Pressman, 2010), (Olszewska, 2019b)</td>
</tr>
</tbody>
</table>

2.1 ONTOLOGY PURPOSE

The scope of this AI-T domain ontology is to assist testers with the testing of AI-based softwares. Indeed, software testing originates from professional practices that have led to a variety of testing approaches and generated different nomenclatures of the testing types and techniques. On the other hand, the testing of AI-based softwares is a new area without established models yet (Hutchinson and Mitchell, 2019). Therefore, the AI-T ontology aims to contribute to the elicitation of the Software Testing knowledge and the formalization of concepts for AI-based softwares’ testing.

Since ontologies intrinsically allow to represent knowledge unambiguously, to share information in an interoperable way, and to perform automated reasoning, the AI-T ontology purpose is to contribute to (i) support human tester(s) in managing software testing; (ii) help human testers in testing AI-based softwares; (iii) guide intelligent agent(s) in generating/reusing test cases; (iv) facilitate intelligent agent(s)’ learning about software testing; (v) aid collaborative mixed human-intelligent agent teams in testing software agents.

2.2 ONTOLOGY BUILDING

The knowledge capture consists in the identification of concepts and relations of both Software Testing and Explainable Artificial Intelligence domains as summed up in Table 1.

Hence, AI-T body of knowledge for software engineering and in particular software testing comprises the SWEBOK guide (Bourque and Fairley, 2014), the ISTQB terminology (ISTQB, 2020), and software engineering standards such as IEEE 24765-2017 (IEEE, 2017) for the software testing vocabulary, IEEE 29119-1-2013 (IEEE, 2013a) for software testing concepts and definitions, IEEE 29119-2-2013 (IEEE, 2013b) for the software testing test processes, IEEE 29119-3-2013 (IEEE, 2013c) for software testing test documentation, IEEE29119-4-2015 (IEEE, 2015b) for software testing test techniques, IEEE29119-5-2016 (IEEE, 2016) for software testing keyword-driven testing, (IEEE, 1994), (Leveson, 2020) for software safety testing as well as books such as (Sommerville, 2015) for software quality testing and (Pressman, 2010) for software measures and metrics.

On the other hand, AI-T body of knowledge for ethical AI general principles (i.e. human rights, well-being, accountability, transparency, awareness
of misuse) includes the IEEE Ethically Aligned Design guidelines (EADv2) (IEEE, 2019), the inter-disciplinary framework for AI ethical practice (e.g. transparency, accountability, privacy, justice, reliability, environmental sustainability) (Hallensleben and Hustedt, 2020), the ethical standards in robotics and AI (Koene et al., 2018b), (Winfield, 2019), (Bryson and Winfield, 2017), (Olszewska et al., 2018), (Olszewska et al., 2020) and subsequent standards such as IEEE P7000 (Spiekermann, 2017), IEEE P7001 for transparency (Wortham et al., 2016), IEEE P7003 (Koene et al., 2018a) for accountability, or IEEE P7010 (IEEE, 2020) for well-being measurement.

The knowledge coding is done in Descriptive Logic (DL) and uses temporal-interval logic relations as introduced in (Olszewska, 2016). Thence, within the Software Testing Process which has been described in Section 1, the concept of ‘Organizational Test Process (OTP)’ is defined in DL, as follows:

\[
\text{Organizational Test Process} \sqsubseteq \text{Software Testing Process}
\]

\[
\begin{align*}
\exists \text{hasProcess}: (\text{OTP} \sqsubseteq \text{DevelopOTP}) \\
\exists \text{hasProcess}: (\text{OTP} \sqsubseteq \text{MonitorOTP}) \\
\exists \text{hasProcess}: (\text{OTP} \sqsubseteq \text{PlanOTP})
\end{align*}
\]  

(1)

with \(OTSn\), an organizational test specification. Furthermore, the OTP processes could be formalised in temporal DL, as follows:

\[
\text{OTPImplementation} \sqsubseteq \text{Organizational Test Process}
\]

\[
\begin{align*}
\exists (\sigma_1, \ldots, \sigma_n) \\
(\text{OTP}_m \sqsubseteq \text{OTP}_l \sqsubseteq \ldots \sqsubseteq \text{OTP}_3 \sqsubseteq \text{OTP}_2 \sqsubseteq \text{OTP}_1)
\end{align*}
\]  

(2)

with \(before\) and \(meet\), the temporal-interval relations as defined respectively in temporal DL:

\[
P_1 < P_2 = before(P_1 \sqsubseteq t_1, P_2 \sqsupseteq t_2) \sqsubseteq \text{TemporalRelation}
\]

\[
\begin{align*}
\exists (\sigma_1, \ldots, \sigma_1) \\
(t_2 < t_1) \\
(P_1 \sqsubseteq t_1, P_2 \sqsupseteq t_2)
\end{align*}
\]  

(3)

\[
P_mP_l = meet(P_l \sqsubseteq t_l, P_m \sqsupseteq t_m) \sqsubseteq \text{TemporalRelation}
\]

\[
\begin{align*}
\exists (\sigma_1, \ldots, \sigma_1) \\
(t_m = t_l) \\
(P_m \sqsubseteq t_m, P_l \sqsupseteq t_l)
\end{align*}
\]  

(4)

where the temporal DL symbol \(\sigma\) represents the temporal existential qualifier, and where a time interval is an ordered set of points \(T = \{t\} \) defined by endpoints \(t^-\) and \(t^+\), such as \( (t^-, t^+) : (t \in T) (t > t^-) \wedge (t < t^+)\).

The concept of ‘Test Management Process (TMP)’ could be represented in DL, as follows:

\[
\text{Test Management Process} \sqsubseteq \text{Software Testing Process}
\]

\[
\begin{align*}
\exists \text{hasProcess}: (\text{TMP} \sqsubseteq \text{TestPlan}) \\
\exists \text{hasProcess}: (\text{TMP} \sqsubseteq \text{TestMonitor and Control}) \\
\exists \text{hasProcess}: (\text{TMP} \sqsubseteq \text{TestCompletion})
\end{align*}
\]  

(5)

Moreover, the TMP processes could be formalised in temporal DL, as follows:

\[
\text{TMPImplementation} \sqsubseteq \text{Test Management Process}
\]

\[
\begin{align*}
\exists (\sigma_1, \ldots, \sigma_1) \\
(\text{TMP}_m \sqsubseteq \text{TMP}_l \sqsubseteq \ldots \sqsubseteq \text{TMP}_3 \sqsubseteq \text{TMP}_2 \sqsubseteq \text{TMP}_1)
\end{align*}
\]  

(6)

The concept of ‘Dynamic Test Process (DTP)’ could be represented in DL, as follows:

\[
\text{DTPImplementation} \sqsubseteq \text{Dynamic Test Process}
\]

\[
\begin{align*}
\exists \text{hasProcess}: (\text{DTP} \sqsubseteq \text{TestDesign and Implementation}) \\
\exists \text{hasProcess}: (\text{DTP} \sqsubseteq \text{TestEnvironment Setup and Maintenance}) \\
\exists \text{hasProcess}: (\text{DTP} \sqsubseteq \text{TestEvaluation}) \\
\exists \text{hasProcess}: (\text{DTP} \sqsubseteq \text{TestResult Reporting})
\end{align*}
\]  

(7)

The DTP processes could be also formalised in temporal DL, as follows:

\[
\text{DTPImplementation} \sqsubseteq \text{Dynamic Test Process}
\]

\[
\begin{align*}
\exists (\sigma_1, \ldots, \sigma_1) \\
(\text{DTP}_m \sqsubseteq \text{DTP}_l \sqsubseteq \ldots \sqsubseteq \text{DTP}_3 \sqsubseteq \text{DTP}_2 \sqsubseteq \text{DTP}_1)
\end{align*}
\]  

(8)

It is worth noting that the temporal-interval relations \(before\) and \(meet\) in Eq. 6 and Eq. 8 are as defined in Eqs. 3-4, respectively.
On the other hand, ethical AI properties such as hasWellBeingTesting could be formalized in DL, as follows:

\[
\text{hasWellBeingTesting} \sqsubseteq \text{Ethical Testing Property} \\
\land \exists \text{hasWellBeing} = \{W_{BI_1}^i\} \\
\land \exists \text{hasWellBeing} = \{W_{BI_1}^j\} \\
\land \exists \text{hasWellBeingValue} = \{W_{BI_1}^j\} \geq \{W_{BI_1}^i\}.
\]

with $W_{BI_1}$, a well-being indicator before using the AI-based software and $W_{BI_1}$, this well-being indicator after using the AI-based software.

The ontology is implemented in the Web Ontology Language (OWL) language, which is the language of all the software testing ontologies (Ferreira de Souza et al., 2013a), and uses Protege v4.0.2 Integrated Development Environment (IDE) with the inbuilt FaCT++ v1.3.0 reasoner (Tsarkov, 2014) to check the internal consistency and to perform automated reasoning on the terms and axioms. An excerpt of the encoded concepts is presented in Fig. 1, while some properties are shown in Fig. 2.

3 VALIDATION AND DISCUSSION

The developed AI-T ontology has been evaluated both quantitatively and qualitatively in a series of experiments as described in Sections 3.1, while its documentation is mentioned in 3.2.

3.1 ONTOLOGY EVALUATION

The ontology evaluation ensures that the developed ontology meets all the requirements (Dobson et al., 2005).

For this purpose, experiments have been carried out using Protege v4.0.2 IDE and applying FaCT++ v1.3.0 reasoner. In particular, an example including the mathematical model and OWL implementation has been provided for testers to get automated guidance during the management process through running AI-T ontology (Fig. 1). On the other hand, an example of testers being supported in testing AI-based softwares, e.g. for the well-being assessment, has been formalized in Section 2.2 and illustrated in Fig. 2, leading to patterns for reusing test cases and facilitating the software testing learning and interoperable knowledge sharing. In qll these experiments, AI-T ontology provided 100% correct answers, and no inconsistency has been observed.

The evaluation of AI-T used metrics such as presented in (Tartir et al., 2018) and (Hlomani and Stacey, 2014). The computed values by Protege are presented in Fig. 3. We can observe that the DL expressivity is $\mathcal{SRQIOIQ(D)}$. It is worth noting the reasoner works under the open-world assumption, i.e. if for a question, there is no information in the ontology, then the answer of the system is 'does not know', and not 'does not exist'. To obtain the latter one, information should be explicitly provided, but adding all these closure-type assertions can slow down the reasoner. So, in practice, a trade-off should be achieved between computational efficiency and completeness. Actually, AI-T performs in real time and contains a total of 707 terms and 705 axioms. This is much higher compared to other state-of-art software testing ontologies that own in average $61 \pm 44$ terms (Tebes et al., 2019). Moreover, AI-T cohesion could be assessed using the number of root classes which is equal to 1, the number of leaf classes which is equal to 521, and the average depth which is equal to 4. All these measures indicate AI-T shows promising performance for real-world deployment.

3.2 ONTOLOGY DOCUMENTATION

The AI-T ontology has been documented in Section 2. It is a middle-out, domain ontology which has been developed from scratch using non-ontological resources such as primary sources, e.g. IEEE standards, as recapitulated in Table 1. AI-T is not dependent of any particular software/system/agent/service/application/project, but it is rather focused on the software testing knowledge as well as the AI system’s ethical principles.

It is worth noting the AI-T has not reused any existing software testing ontology, and it is the first ontology in its kind for the AI-software’s testing domain. Moreover, the AI-T ontology could be used in conjunction with other ontologies such as the core ontology for autonomous systems (CORA) (IEEE, 2015a) or other robotics and automation ontologies (Fiorini et al., 2017) for further integration.
4 CONCLUSIONS

With the increasing usage of artificial intelligence (AI) and especially machine learning (ML) within softwares as well as the growing number of autonomous intelligent agents acting outside systems, the effective testing of AI-based softwares is crucial and should have both a wide test coverage and a rigorous formalization. Hence, we propose an ontological approach for testing AI-based softwares. In particular, we proposed a new domain ontology which is called AI-T and which encompasses software testing knowledge and ethical AI principles. AI-T purpose is to guide AI-based softwares’ testing, to automate their testing, and also facilitate the learning about software testing. AI-T ontology has both conceptual and implementation models that are helpful for the interoperable sharing of the AI-based software testing terms and relations, and that have been successfully applied for whatever manual or automated testing.

REFERENCES


ISTQB (2020). International software testing qualifications board.


