Is Agent Software More Complex than Other Software?

Extended Abstract

Alon Zanbar
The MAVERICK Group
Bar Ilan University
Ramat Gan, Israel
atzanbar@gmail.com

Gal A. Kaminka
The MAVERICK Group
Bar Ilan University
Ramat Gan, Israel
galk@cs.biu.ac.il

ABSTRACT

We empirically investigate agent software repositories using commonly used software metrics, which are used in software engineering literature to quantify meaningful characteristics of software based on its source code. We contrast the measurements with those of software in other categories. Analyzing hundreds of software projects, we find that agent software may be different from other types of software, in terms of software complexity measures.

KEYWORDS

Agent-Oriented Software Engineering; Software Metrics; AI and Software Engineering

1 INTRODUCTION

For many years, significant research efforts have been spent on investigating methodologies, tools, models and technologies for engineering autonomous agents software. Research into agent architectures and their structure, programming languages specialized for building agents, formal models and their implementation, development methodologies, middleware software, have been discussed in the literature, encompassing multiple communities of researchers, with at least partial overlaps in interests and approaches.

The most important underlying assumption of these research efforts is that such specialization is *needed*, because autonomous agent software poses engineering requirements that may not be easily met by more general (and more familiar) software engineering and programming paradigms. Specialized tools, models, programming languages, code architectures and abstractions make sense, if the software engineering problem is specialized.

This paper provides *the first empirical evidence for the distinctiveness of autonomous agent software*, compared to other software categories. We utilize basic source code metrics, such as *Cyclomatic Complexity*, *Cohesion*, *Coupling*, and others. These metrics are commonly used by researchers and practitioners to assess code quality, estimate work effort, and to quantify other meaningful characteristics of software. We use them to understand how agent and robot software is different from general software.

We quantitatively contrast close to 140 autonomous agent and robot projects (from RoboCup, The Agent Negotiations Competitions, Chess, and various robotics projects), with close to 400 other software projects from github, of various types. Each was quantified using over 250 metrics. We then conducted both manual analysis and automated machine-learning analysis of the differences between agent software, robot software, and other software.

We find that agent software is clearly and significantly different from other types of software of comparable size. This result appears both when using manual statistical analysis, as well as machine learning methods. Specifically, autonomous agents software is significantly more complex (in the sense of control flow complexity) than other software categories. We discuss potential implications of these results.

2 BACKGROUND

There is vast literature reporting on research that directly or indirectly impacts software engineering and development of autonomous agents: AOSE agent-oriented software engineering is a thriving area of research, with at least one dedicated annual conference/workshop and a specialized journal[1] [1–5, 7–11, 13–24]. For the most part, the arguments for the study of AOSE as distinct from general software engineering are well argued *philosophically*, and *qualitatively* pointing out inherent conceptual differences between the software engineering of agents, and other software domains. To the best of our knowledge, no empirical evidence—certainly not at the scale detailed below—has been offered to support these important conceptual arguments.

The lack of quantitative investigations of agent and robot software engineering is not for lack of quantitative and empirical methods in software engineering in general. Many investigations—starting in the early seventies and continuing today—propose quantitative metrics of software constructs, and relate the measurements to software quality, development effort, software type, and other attributes of interest. Comprehensive reviews of these are presented in [6, 12] for example.

3 SOFTWARE PROJECT DATA COLLECTION AND CURATION

We are looking to compare general software categories to software implementing autonomous agents operating in virtual and physical environments (i.e., robots). We begin with an overview of the data collection and curation process, in preparation for the manual and automated analysis described in the following sections.

We use several sources of software projects, each containing multiple projects:

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

[1] International Journal of Agent-Oriented Software Engineering

RoboCup. RoboCup is the among the largest annual global robotics competition events in the world, and one of the longest running—taking place since 1997 (1996 pre-competition).

ANAC.. The annual International Automated Negotiating Agents Competition (ANAC) is used by the automated negotiation research community to benchmark and evaluate its work and to challenge itself.

GitHub. GitHub, the largest repository of open source projects in the world. The categories we select are distinct as much as possible from AI code and from one another.

From the sources above, we collected software projects that meet maturity and size criteria, and are easily identified as belong to specific software categories. Those project where analyzed using open source static code analysois tools to produce large list of software metrics. the following list demonstrate some of them :

Summary & code Metrics: Total Lines of Code (total_loc) , Total Number of Modules (total_modules) , Total Number of Methods (total_nom) , Afferent Connections per Class (ACC) , Average Cyclomatic Complexity per Method (ACCM) , Average Method Lines of Code (AMLOC) , Average Number of Parameters (ANPM) , Coupling Between Objects (CBO) , Coupling Factor (COF) , Depth of Inheritance Tree (DIT)

4 MANUAL ANALYSIS

We conducted two separate analysis efforts which had common general goal. a statistical analysis and machine-learning analysis. The focus in both is to reveal differences, if they occur, between the different software categories, as expressed in the measurements of different metrics.

In the statistical analysis we used a heuristic procedure to assist in finding promising features. The idea is to iterate over the software domains. For each domain r, we separate it out from the others, and then use a two-tailed t-test to contrast the distribution of the metric values in the domain and in all others. We use the p value generated by the above procedure to form clusters of three or more software domain that share similar p value in the same metrics. Those groups were used as indicator for possible clusters. by ordering the increasingly by p we could see the four group with lowest common p value are those containing the results of testing ACCM values of "Agent" types against other software categories.

Visualization of box-plots distributions of specific metrics of each software domain, revealed some differences between the software categories. We discovered that the most noticeable results are of some complexity metrics as ACCM and MLOC. The distribution of those metrics for "Agent" software shows higher values than in all other domains.

5 MACHINE LEARNING ANALYSIS

A second approach for our investigation uses machine learning techniques, to complement the manual analysis. We attempted to use several different machine learning classifiers to distinguish agent and non-agent software domains, with the goal of analyzing successful classification schemes, to reveal the metrics, or metric combinations, which prove meaningful in the classification

Classification procedure. We choose one vs many classification strategy, similarly to the manual analysis above. Iterating over all software classes, we trained a binary classifier to differentiate between samples of one software domain (for example, Audio) to all other software classes. This creates an inherent imbalance in the number of examples presented, which we alleviated by using random over-sampling of the minority class.

For classification, we used the following classification algorithms: *Support Vector Machines*, *Logistics Regression*, and *Gradient-Boosted Decision Trees*. The implementations are open-source. The performance of classifiers was carried out using two scoring functions, familiar to machine learning practitioners: F1 and AUC (area under the ROC curve).

The results implies that the top performing classifiers (1) are those that are able to distinguish agent software from other types of software, and (2) utilize the mean ACCM and AMLOC metrics in their classification decisions. These results concur with the conclusions of the manual analysis described earlier.

Table 1 reflects the performance of each XGBoost classifiers trained to separate between the different software classes. The table demonstrate the superior results of separating "Agent" repositories compared to other software domains. A complementary list of features that are most significant for the XGBoost model for classifying "Agent" repositories against other domains is presented in Table 2:

	Agent/General	Class (Domain)	AUC	F1
0	Agent	Robocup-2D	0.97	0.85
1	Agent	ANAC	0.98	0.67
2	Agent	Chess	0.84	0.44
3	Robot	Robcup-Other-Leagues	0.89	0.40
4	General	Graphics	0.65	0.31
5	General	Security	0.76	0.27
6	General	Mobile	0.80	0.22
7	General	Games	0.49	0.00
8	General	Audio	0.56	0.00
9	General	Robot-Simulation	0.66	0.00
10	General	Education	0.66	0.00
11	General	Finance	0.73	0.00
12	General	IDE	0.75	0.00
13	Robot	Robo-Projects	0.86	0.00

Table 1: Gradient Boosted Decision Trees top scoring software classes. Mean ACCM is a recurring important feature.

Class (Domain)	Important Features
Robocup-2D	[amloc_mean, mmloc_mean, noc_mean, rfc_mean]
ANAC	[accm_mean, noa_mean, npa_mean, npm_mean]
Chess	[accm_mean, cbo_mean, dit_mean, lcom4_mean]

Table 2: XGBoost important features for classifying Agent repositories

Acknowledgments. This research was supported in part by ISF Grant #2306/18. As always, thanks to K. Ushi.

REFERENCES

[1] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, and Alessandro Ricci. 2014. Unravelling Multi-agent-Oriented Programming. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 259–272. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_13

[2] Mehdi Dastani. 2014. A Survey of Multi-agent Programming Languages and Frameworks. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 213–233. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_11

[3] Christian Detweiler, Koen Hindriks, and Catholijn Jonker. 2010. Principles for Value-Sensitive Agent-Oriented Software Engineering. In *Agent-Oriented Software Engineering XI (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg.

[4] Avshalom Elmalech, David Sarne, and Noa Agmon. 2014. Can Agent Development Affect Developer's Strategy?. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[5] Amir Elmishali, Roni Stern, and Meir Kalech. 2016. Data-Augmented Software Diagnosis. In *Innovative Applications of AI (IAAI)*.

[6] Norman Fenton and James Bieman. 2014-10-01. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. CRC Press. Google-Books-ID: lx_OBQAAQBAJ.

[7] Klaus Fischer, Christian Hahn, and Cristian Madrigal Mora. 2007. Agent-oriented software engineering: a model-driven approach. *International Journal of Agent-Oriented Software Engineering* 1, 3/4 (2007), 334. http://www.inderscience.com/link.php?id=16265

[8] Koen V. Hindriks and Jügen Dix. 2014. GOAL: A Multi-agent Programming Language Applied to an Exploration Game. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 235–258. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_12

[9] Marc-Philippe Huget. 2014. Agent Communication. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 101–133. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_6

[10] Nicholas R. Jennings. 2000-03-01. On agent-based software engineering. *Artificial Intelligence* 117, 2 (2000-03-01), 277–296. http://www.sciencedirect.com/science/article/pii/S0004370299001071

[11] Nicholas R. Jennings. 2001. An agent-based approach for building complex software systems. *Commun. ACM* 44, 4 (April 2001), 35–41. http://portal.acm.org/citation.cfm?doid=367211.367250

[12] Capers Jones. 2008. *Applied Software Measurement: Global Analysis of Productivity and Quality* (3rd ed.). McGraw-Hill, New York.

[13] Joanna Juziuk, Danny Weyns, and Tom Holvoet. 2014. Design Patterns for Multi-agent Systems: A Systematic Literature Review. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 79–99. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_5

[14] Renato Levy and Goutam Satapathy. 2014. Design and Implementation of Very Large Agent-Based Systems. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 289–307. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_15

[15] Ashok U. Mallya and Munindar P. Singh. 2006. Incorporating Commitment Protocols into Tropos. In *Agent-Oriented Software Engineering VI*, Jörg P. Müller and Franco Zambonelli (Eds.). Vol. 3950. Springer Berlin Heidelberg, Berlin, Heidelberg, 69–80. http://link.springer.com/10.1007/11752660_6

[16] Jörg P. Müller and Klaus Fischer. 2014. Application Impact of Multi-agent Systems and Technologies: A Survey. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 27–53. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_3

[17] Lin Padgham, John Thangarajah, and Michael Winikoff. 2014. Prometheus Research Directions. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 155–171. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_8

[18] Lin Padgham and Michael Winikoff. 2004. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons, Ltd, Chichester, UK. http://doi.wiley.com/10.1002/0470861223

[19] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. 2008. An architecture for exception management in multiagent systems. *International Journal of Agent-Oriented Software Engineering* 2, 3 (2008), 267. http://www.inderscience.com/link.php?id=19420

[20] Yoav Shoham. 1991. Agent-oriented programming. *Artificial Intelligence* 60 (1991), 51–92.

[21] Arnon Sturm and Onn Shehory. 2014. Agent-Oriented Software Engineering: Revisiting the State of the Art. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, 13–26. https://link.springer.com/chapter/10.1007/978-3-642-54432-3_2

[22] P. R. Telang and M. P. Singh. 2012. Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach. *IEEE Transactions on Services Computing* 5, 3 (2012), 305–318.

[23] Michael Winikoff. 2009. Future Directions for Agent-Based Software Engineering. *International Journal of Agent-Oriented Software Engineering* 3, 4 (May 2009), 402–410. http://dx.doi.org/10.1504/IJAOSE.2009.025319

[24] Ari Yakir and Gal A. Kaminka. 2007. An Integrated Development Environment and Architecture for Soar-Based Agents. In *Innovative Applications of Artificial Intelligence (IAAI-07)*.