Empirical Software Engineering for Agent Programming

M. Birna van Riemsdijk

Interactive Intelligence
Delft University of Technology
Mekelweg 4, 2628 CD, Delft
The Netherlands
m.b.vanriemsdijk@tudelft.nl

Abstract

Empirical software engineering is a branch of software engineering in which empirical methods are used to evaluate and develop tools, languages and techniques. In this position paper we argue for the use of empirical methods to advance the area of agent programming. Through that we will complement the solid theoretical foundations of the field with a thorough understanding of how our languages and platforms are used in practice, what the main problems and effective solutions are, and how to improve our technology based on empirical findings. Ultimately, this will pave the way for establishing multi-agent systems as a mature and recognized software engineering paradigm with clearly identified advantages and application domains.

Categories and Subject Descriptors I.2.5 [*Artificial Intelligence*]: Programming Languages and Software; I.2.11 [*Artificial Intelligence*]: Distributed Artificial Intelligence—Intelligent agents, languages and structures, Multiagent systems; D.2 [*Software Engineering*]

General Terms Design, Languages

Keywords Agent programming languages, empirical software engineering, software quality, metrics

1. Introduction

Empirical software engineering is a branch of software engineering in which empirical methods are used to evaluate and develop tools, languages and techniques. The journal on Empirical Software Engineering (see [1]) started in 1996. As stated in [14]: 'The acceptance of empirical studies in software engineering and their contributions to increasing knowledge is continuously growing. The analytical research paradigm is not sufficient for investigating complex real life issues, involving humans and their interactions with technology.' That is, empirical research needs to complement theoretical studies in order to advance understanding with respect to the use of technologies.

We argue that empirical software engineering is important not only for mainstream software engineering, but also for agent-oriented programming [3, 4] and software engineering. Through empirical methods, different kinds of questions can be answered than through analytical approaches. In this position paper we propose several such questions and thereby sketch what such a line of research might look like. We focus on agent programming rather than agent-oriented software engineering in general. However, similar questions and issues as proposed below for agent programming may also be translated to agent-oriented software engineering. The term 'agent programming' should be understood to refer to programming autonomous agents and multi-agent systems.

2. Research Questions

By using empirical methods, data can be gathered for several purposes. For example, to get a better understanding of how software developers use agent programming technology (problems and possible solution patterns), to perform a within technology comparison, i.e., demonstrating that one variant of (using) agent programming technology is better than another, to improve the technology based on these findings, and to perform across technology comparison, i.e., demonstrating that agent programming technology is better than some other technology. Corresponding research questions that may be studied using empirical methods can concern any of the set of instruments that facilitate the development of high-quality agent programs, namely programming language, programming guidelines & teaching methods, and development environment.

AGERE! 2012, October 21–22, 2012, Tucson, Arizona, USA.

For example:

- how do programmers use agent-oriented languages?
  - which constructs do they use?
  - what is expressed using which constructs?
  - what kind of patterns do they use?
  - which problems do they experience while programming?
  - which aspects of the languages do they find difficult or easy to understand?
  - what kind of processes are used during development, e.g., which parts of a program are developed first?
- which ways of using agent-oriented languages improve agent software quality?
  - how should constructs be used?
  - which patterns and anti-patterns can be distinguished?
- how does the use of an agent programming language compare with the use of mainstream, general purpose languages like Java, and other paradigms for development of decentralized, concurrent applications?
  - do similar programming patterns emerge?
  - how does the speed of software development compare?
  - how do the resulting programs compare with respect to software quality measures like efficiency, maintainability and readability?
- how does the domain for which applications are developed, influence software development?
  - which domain characteristics call for an agent-oriented approach to software development?
  - which patterns are/can/should be used in which kinds of domains?
- which features should an Integrated Development Environment for agent programming have?
  - how do programmers use existing IDEs?
  - which difficulties do they encounter while programming?
  - to what extent do requirements for an IDE for agent-oriented software development differ from those for mainstream software development?
  - which approaches for debugging are needed in the context of agent programming?
  - to what extent does debugging in agent programming differ from debugging in mainstream software development?

Of course, several of these questions have already been addressed to some extent in various papers. For example, in [2] it is shown that the use of BDI technology incorporated within an enterprise-level architecture can improve overall developer productivity by an average 350%. They argue that agent technology is particularly suitable for applications that are "hard" to build, in which requirements change quickly and which are event and exception driven. Testing multi-agent systems has also been studied in several papers, e.g., [11, 13, 17], although only [13] reports some empirical results. In [6], an empirical study is performed in the area of game development, where the POSH reactive planner with a graphical editor is compared with Java for programming high-level behavior of a virtual agent in the Unreal Tournament 2004 environment. In [9], metrics for quantifying coupling and cohesion are proposed that can be applied to agents as well as object-oriented software. In [20] (which is based on [19] and [8]) we have studied how programmers use the GOAL agent programming language, making several observations concerning, e.g., the use of programming constructs and patterns. Similarly, in [12] the authors report on experiences in using the Jason agent programming language by a novice Jason programmer. They suggest programming patterns to address several encountered issues.

3. Software Quality

A recurring theme in the above research questions is *software quality*. We aim at developing techniques that facilitate building "better" software. The question is then what exactly we mean by better software, i.e., how do we define software quality? A starting point for this is the ISO/EIC 9126 standard which provides a software quality model. It defines several software quality characteristics and subcharacteristics: functionality (e.g., interoperability, functionality compliance), reliability (e.g., fault tolerance, recoverability), usability (e.g., understandability, learnability), efficiency (e.g., time behavior), maintainability (e.g., analyzability, testability), and portability (e.g., adaptability, co-existence).

These characteristics provide an indication of what kind of characteristics to address when aiming for better software, but they do not specify how to *measure* to what extent a certain piece of software exhibits a characteristic. To address this, the standard specifies that for each characteristic a set of attributes has to be defined that can be verified or measured in the software product. This can be done, for example by defining a set of *quality metrics* which evaluate the degree of presence of quality attributes in the software. These can be internal metrics (static), external metrics (defined for running software), or 'quality in use' metrics (defined for using the software in real conditions). These attributes and metrics vary between technologies and software products.

Research will have to identify what software quality means in the context of programming multi-agent systems (MAS). Questions that need to be addressed are:

- Which ISO software quality characteristics are suitable for MAS?
- Which ISO software quality characteristics are particularly important (problematic or strength) in MAS?
- Which are MAS-specific software quality characteristics?
- Which MAS-specific attributes and metrics can be defined for measuring the characteristics?

It will be interesting to make precise how to measure quality characteristics in MAS. Given the wide range of languages and platforms for programming MAS, it should be of particular concern to analyze to what extent language-independent measures can be defined (as in [9]), or whether certain quality metrics need to be defined specific to a particular technology. For example, in [15] several existing software engineering metrics are used to evaluate a methodology for creating affective applications. In [16] metrics are used for evaluating the quality of message sequence charts, in the context of evaluating a methodology for developing cross-organizational business models. It will have to be investigated how to compare agent-specific metrics for quantifying a certain quality characteristic with metrics for that characteristic in mainstream technologies. Finally, it will be interesting to identify quality characteristics that are specific to MAS. Examples of characteristics that may be considered are *explainability* (to what extent is the intelligent system able to explain its decisions; this can be important for acceptance of the technology and for debugging (see, e.g., [7, 18])), and *believability* (to what extent does an intelligent (virtual) character or group of characters display believable behavior; this can be important for example for creating natural interaction with a human user of the technology).

4. Methodological Aspects

In this paper we argue for recognition of a line of research on empirical software engineering for agent programming, and for a more systematic approach to addressing questions like those posed above. This also calls for discussion and research concerning appropriate *methodologies* for conducting empirical research in agent programming. It needs to be investigated whether methods from mainstream empirical software engineering can be applied in our context. For example, [14] proposes guidelines for conducting and reporting case study research in software engineering. In [20] we propose an approach for empirically studying how programmers use an agent programming language, in which we identify several analysis dimensions, such as a functional analysis which identifies what the available language constructs are used for, and which general principles are applied when using them; and a structural analysis which identifies structural code patterns, and which determines quantitative metrics on the code. Also we propose a step-wise research approach for conducting case study research in agent programming, which is based on [5].

We believe that both quantitative as well as qualitative research should be performed. Quantitative research is used for testing pre-determined hypotheses and producing generalizable results using statistics, focusing on answering mechanistic 'what?' questions; for example, what is the effect of using a certain debugging tool on the number of errors in the resulting software. Qualitative research is used for illumination and understanding of complex psychosocial issues, and can be used for answering humanistic 'why?' and 'how?' questions [10]; for example, how do programmers use agent-oriented programming languages. We believe that in particular in the earlier stages of studying the use of agent programming language empirically, it is very important to also perform qualitative research. This will provide a better understanding of how they are used, and through this the techniques can be improved. Once sufficient improvement has been realized through this process, within and across technology comparisons can be performed in a quantitative manner.

5. Conclusion

In this position paper we have argued for the use of empirical methods to advance the area of agent programming. Through this we will complement the solid theoretical foundations of the field of agent programming with a thorough understanding of how our languages and platforms are used practice, what the main problems and effective solutions are, and how to improve our technology based on empirical findings. Ultimately, this will pave the way for establishing multi-agent systems as a mature and recognized software engineering paradigm with clearly identified advantages and application domains.

Acknowledgments

I would like to thank Koen V. Hindriks for the joint research and many discussions on the theme of this position paper. Also, I would like to thank the organizers (Jürgen Dix, Koen V. Hindriks, Brian Logan and Wayne Wobcke) and participants of the Dagstuhl Seminar 12342 on Engineering Multi-Agent Systems which was held in August 2012 for giving me a chance to present these ideas and discuss them with the audience.

References

[1] V. R. Basili and L. C. Briand, editors. *Empirical Software Engineering: An International Journal*. Springer, 2012. `http://www.springer.com/computer/swe/journal/10664`.

[2] S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS'06)*, pages 10–15. ACM, 2006.

[3] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.

[4] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Berlin, 2009.

[5] K. Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, 1989.

[6] J. Gemrot, Z. Hlávka, and C. Brom. Does high-level behavior specification tool make production of virtual agent behaviors better? In *Proceedings of the International Workshop on Cognitive Agents for Virtual Environments (CAVE'12)*, 2012.

[7] K. V. Hindriks. Debugging is explaining. In *Principles of Practice in Multi-Agent Systems (PRIMA'12)*, volume 7455 of *LNAI*, pages 31–45. Springer, 2012.

[8] K. V. Hindriks, M. B. van Riemsdijk, and C. M. Jonker. An empirical study of patterns in agent programs. In *Principles of Practice in Multi-Agent Systems (PRIMA'10)*, volume 7057 of *LNAI*, pages 196–211. Springer, 2011. Best paper award (runner up).

[9] H. R. Jordan and R. Collier. Evaluating agent-oriented programs: Towards multi-paradigm metrics. In *Proceedings of the Eigth International Workshop on Programming Multiagent Systems (ProMAS'10)*, volume 6599 of *LNCS*, pages 63–78. Springer, 2012.

[10] M. N. Marshall. Sampling for qualitative research. *Family Practice*, (3):522–525, 1996.

[11] S. Miles, M. Winikoff, S. Cranefield, C. D. Nguyen, A. Perini, P. Tonella, M. Harman, and M. Luck. Why testing autonomous agents is hard and what can be done about it. URL `http://www.pa.icar.cnr.it/cossentino/AOSETF10/docs/miles.pdf`. AOSE Technical Forum 2010 Working Paper.

[12] R. Píbil, P. Novák, C. Brom, and J. Gemrot. Notes on pragmatic agent-programming with jason. In *Proceedings of the Nineth International Workshop on Programming Multiagent Systems (ProMAS'11)*, volume 7217 of *LNCS*, pages 58–73. Springer, 2012.

[13] D. Poutakidis, M. Winikoff, L. Padgham, and Z. Zhang. Debugging and testing of multi-agent systems using design artefacts. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 215–258. Springer, Berlin, 2009.

[14] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[15] D. J. Sollenberger and M. P. Singh. Kokomo: an empirically evaluated methodology for affective applications. In *Proceedings of the tenth international joint conference on autonomous agents and multiagent systems (AAMAS'11)*, pages 293–300. IFAAMAS, 2011.

[16] P. R. Telang and M. P. Singh. Comma: a commitment-based business modeling methodology and its empirical evaluation. In *Proceedings of the eleventh international joint conference on autonomous agents and multiagent systems (AAMAS'12)*, pages 1073–1080. IFAAMAS, 2012.

[17] J. Thangarajah, G. Jayatilleke, and L. Padgham. Scenarios for system requirements traceability and testing. In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, pages 285–292. IFAAMAS, 2011.

[18] I. van de Kieft, C. M. Jonker, and M. B. van Riemsdijk. Explaining negotiation: Obtaining a shared mental model of preferences. In *24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE'11)*, volume 6704 of *LNCS*, pages 120–129. Springer, 2011.

[19] M. B. van Riemsdijk and K. V. Hindriks. An empirical study of agent programs: A dynamic blocks world case study in GOAL. In J.-J. Yang, M. Yokoo, T. Ito, Z. Jin, and P. Scerri, editors, *Principles of Practice in Multi-Agent Systems*, volume 5925 of *LNAI*, pages 200–215. Springer, 2009. Best paper award.

[20] M. B. van Riemsdijk, K. V. Hindriks, and C. M. Jonker. An empirical study of cognitive agent programs. *Multiagent and Grid Systems (MAGS)*, 8(2):187–222, 2012.