World Transactions on Engineering and Technology Education
Vol.2, No.2, 2003

Agent-Oriented Software Engineering (AOSE): its emergence as a cornerstone of enterprise software engineering education

Gilda Pour

San José State University
San José, United States of America

ABSTRACT: There is a fast growing demand for a wide variety of enterprise and mission-critical applications including computing, communications, space, air traffic control systems, defence, health care, manufacturing, finance, highway safety, transportation (eg the Intelligent Vehicle Highway Systems), power generation and transmission, e-commerce and any others affecting human safety or well-being. Many of those systems are required to be autonomous, extensible, flexible, robust, reliable and capable of being remotely monitored and controlled. This imposes a key challenge on developing enterprise and mission-critical software systems. Agent-Oriented Software Engineering (AOSE) has emerged as a viable approach to address this challenge, and also as a cornerstone of enterprise software engineering. The article discusses the critical need for restructuring enterprise software engineering education by integrating research and education in AOSE and by building a foundation for life-long learning. The article also presents a strategy for restructuring software engineering education to help narrow the gap between what the vast majority of software engineering education programmes offer and what the real world expects of graduates of software engineering education programmes.

INTRODUCTION

Demand for an extensive selection of enterprise and mission-critical applications, including computing, communications, space, nation's air traffic control system, defence, health care, manufacturing, finance, highway safety, transportation (eg the Intelligent Vehicle Highway Systems), power generation and transmission, e-commerce, as well as any others that affect human safety or well-being, is growing rapidly. This has motivated the search for new software engineering methodologies and development strategies.

These new strategies must support the development of enterprise software applications that are autonomous, extensible, robust, reliable and capable of being remotely monitored and controlled. Traditional software engineering falls short in this regard. Agent-Oriented Software Engineering (AOSE) has emerged as an attractive alternative for building enterprise and mission-critical applications; it has also developed as a cornerstone for enterprise software engineering.

AGENT-ORIENTED SOFTWARE ENGINEERING

Agent-Oriented Software Engineering (AOSE) extends Component-Based Software Engineering (CBSE). AOSE also leads to greater flexibility, adaptability and autonomy [1-4]. AOSE, much like CBSE, has great potential to reduce enterprise software development costs and time-to-market, while also improving reliability, maintainability and overall quality of enterprise software systems [1][5][6]. AOSE is based on developing and evolving software systems from selected pre-engineered and pre-tested software agent components.

Software agent components (a.k.a. agents) are viewed as next-generation software components (specialised distributed components), which offer greater flexibility, adaptability and autonomy than traditional software components [1]. An agent is also viewed as an autonomous entity driven by a set of Beliefs, Desires and Intentions (a.k.a. BDI). Agent components can exchange messages and operate based on their beliefs and goals; they are not necessarily implemented using Artificial Intelligence (AI) technology.

An agent component exhibits a combination of several of the following characteristics:

- Autonomous: being able to proactively initiate activities based upon its goals, to act on the behalf of its user and to exercise control over its actions.
- Adaptable: being able to change its behaviour after deployment, either by its own learning, user customisation or downloading new capabilities.
- Mobile: being able to move from one executing context to another, to continue execution in a new context and to retain its state to continue its work.
- Collaborative: being able to communicate and cooperate with other agents to form dynamic or static societies of agents, collaborating to perform a task.
- Knowledgeable: being able to reason about its goals, acquired information and knowledge about other agents and users.
- Persistence: being able to retain its knowledge and state over an extended time period, system crashes, multiple sessions, etc [1].

An agent can be classified in one of the following categories, depending upon the type of function that the agent performs:

- Collaborative agents: Agents communicating and interacting (eg exchanging messages to negotiate or share information) with some other agents that represent their users, organisations and services.

- Mobile agents: Agents visiting remote sites to collect information before returning with the results, and aggregating and analysing data, or exercising local control.
- Personal agents: Agents interacting directly with a user, presenting some personality or character, monitoring and adapting to its user's activities, learning the user's style and preferences, and automating or simplifying certain tasks [1].

Agent components and infrastructure have great promise in significantly increasing the dependability of enterprise and mission-critical applications, primarily because of certain fundamental elements that are typically associated with agents, such as autonomy, adaptability and collaboration.

Furthermore, agent-oriented software engineering methods and technology provides a basis for engineering autonomic, self-managing, self-healing and self-tuning systems that are able to dynamically adjust and reconfigure themselves in response to changes in their environment.

NEW ROLES AND NEW COMPETENCES

Due to its nature, AOSE requires new software engineering roles and new competences that are significantly different in many ways from those in traditional software engineering.

New software engineering roles within AOSE are classified under the following two domains, namely:

- Agent Component System Engineering (ACSE)
- Multi-Agent System Engineering (MASE)

ACSE deals with the development of agent components, while MASE handles the development of multi-agent systems. Each domain requires a different set of competences. The ACSE lifecycle includes the following major phases:

- Agent component requirements determination and analysis;
- Agent component design;
- Agent component implementation;
- Agent component testing;
- Agent component maintenance and evolution.

The design phase in the ACSE lifecycle deals with the design of agent component systems. The issues that need to be addressed in the design phase depend on the category of the agent. For example, in the design of mobile agents, software engineers need to deal with additional complexity due to security concerns associated with agent mobility.

The MASE lifecycle includes the following phases:

- Multi-agent system requirements determination and analysis;
- Selection and customisation of a set of agents;
- Selection and customisation of multi-agent system architectures;
- Multi-agent system integration;
- Multi-agent system testing;
- Multi-agent system maintenance and evolution.

The design phase in the MASE lifecycle includes the selection and customisation of both agents and multi-agent system architectures. The multi-agent system architecture presents agent components as interacting service provider or consumer entities. It also facilitates agent operations and interactions under environmental constraints, and also allows agents to take advantage of available services and facilities. The design of multi-agent systems must deal with the following aspects:

- Agent-to-agent communications that involve agent's FIPA-complaint messages, conversation protocols, message content, standard/dynamic ontology (agent vocabularies), and coordination between agents and/or groups of agents;
- An agent platform that provides basic capabilities of multi-machine agent transport, agent lifecycle management, and interfaces to application services, such as those provided by the J2EE, .NET or OS;
- Standard service agents that provide capabilities for the support of dynamic agent creation and discovery;
- Agent-to-service interfaces that provide standardised agent interfaces to non-agent capabilities;
- Agent internals that specify the way agents deal with events, communicate among subsystems, set-up and run behaviours autonomously, deal with exceptions, and monitor and react to interactions with other agents.

Multi-agent system integration is the replacement for the implementation phase. While the implementation phase in traditional software engineering requires extensive programming in order to build enterprise systems from scratch, the multi-agent system integration phase involves developing agent wrappers and mediators in order to support interactions among agents within a system and outside of the system.

The multi-agent system testing phase requires the integration testing of a software system that is an assembly of a set of software agents that have been developed by different developers, independent of one another. The challenges faced in system integration testing, maintenance and evolution are mainly due to the lack of confidence in, and understanding of, agents built by other developers [7].

There are also other new roles and competences for dealing with legal issues associated with development and usage of agent-oriented software systems, as well as with the marketing and sales of those systems. Legal issues involve intellectual property law for copyrights and patents, legal liability, licensing, support and warranties. The legal system of intellectual property rights and liabilities profoundly affects the economic viability and course of new technologies [8].

STRATEGIES FOR RESTRUCTURING SOFTWARE ENGINEERING EDUCATION

Due to the major differences between the software engineering roles and competences required in agent-oriented software engineering, and those required in the traditional software engineering, restructuring software engineering education is inevitable. Restructuring efforts need to emphasise the following elements:

- Building a solid foundation for life-long learning;
- Integrating agent-oriented software engineering research and education into the enterprise software engineering curricula;

- Partnering with industry in education and research;
- Integrating diversity into the programme.

Building a Foundation for Life-Long Learning

Software technologies crop up quickly and the role of Information Technology (IT) is constantly changing. This necessitates any software engineering education programme to facilitate the process of building a solid foundation for effective life-long learning for their students [9]. This can be achieved by enabling students to do the following:

- Recognise the crucial need for life-long learning;
- Actively engage in life-long learning throughout their career.

In order to help students recognise the crucial need for life-long learning, the importance and necessity of upgrading their knowledge and skills should be discussed in software engineering classes. In addition, students should study the impact of a rapidly evolving and highly diversified world of enterprise software technology on a software engineering career.

Other factors that need to be accentuated in software engineering education include the following:

- Broader understanding of the impacts of information technology competence on one's career;
- Systematic thinking;
- Cooperative problem solving;
- Ability to communicate effectively;
- Systems view of problem solving;
- Technical and economic decision-making required in enterprise software engineering.

In order to help students acquire the ability to seek, evaluate and use information not directly provided by assigned texts and lectures, the following two main factors should be considered:

- Having a solid foundation of fundamental knowledge and skills;
- Knowing how to acquire and effectively learn new materials on one's own.

The first factor should be well integrated into software engineering curricula with special emphasis on learning the required fundamental concepts, systematic thinking through engineering problems, as well as integrating and applying knowledge so as to solve those problems.

With regard to the second factor, students need to *learn how to learn.* The term, *learn how to learn*, has the following three distinct meanings:

- How to be a better student;
- How to conduct inquiry and construct knowledge in his/her discipline or field;
- How to be a self-directing learner [10].

Table 1 shows the learning objectives for *learn how to learn* and a list of course activities in order to meet each of the learning objectives.

Table 1: Key learning objectives and course activities in order to meet the objectives for *learn how to learn.*

No.	Learning Objectives	Course Activities to Meet the Learning Objectives
1	Students will learn how to be a better student.	• Students are given assignments that require students to seek, read, evaluate and use information and materials beyond the texts and lectures. • Students are required to participate in class discussions and brain-storming practises in small groups. • Students are provided tips on how to get organised for learning on their own and how to engage in active and cooperative learning.
2	Students will learn how to conduct inquiry and construct knowledge in their own discipline or field.	• Students are given assignments that require them to formulate questions and then to work on answering them. • Students learn and practise how to search for and identify relevant information and then analyse that information in order to answer a question or solve a problem. • Students are required to discuss what was learned and how it was learned as a part of their participation in class discussions and activities.
3	Students will learn how to become a self-directing learner.	• Students are given assignments that require them to reflect on their own learning and also prompt them to explore the impact of their own learning processes on how they should teach in the future. • Students are given assignments that require them to do the following: - Develop a learning agenda by engaging in thinking towards the future and identifying what else they need or want to learn. - Develop a plan of action by identifying specific actions for learning the items on their learning agenda. Specific action could be talking to an expert or experienced person, reading a book on the topic, finding information in technical publications, observing or practising something.

In order to help students learn and enhance the skills required for life-long learning, the author recommends engaging students in curricular and extracurricular activities that include the following:

- Predicting the development of future software technologies through various research tools and models.
- Recognising and comparing various approaches in order to solve real-world software engineering problems.
- Recognising the constraints of different approaches.

- Discussing trade-off decision-making with respect to different approaches in order to solve various software engineering problems.
- Learning and enhancing one's effective listening and communication skills, as well as critical reading, thinking and writing skills.

Integrating Research and Education

In order to narrow the gap between what the vast majority of software engineering education programmes offer and what the real world expects of graduates of software engineering education programmes, it is essential to integrate AOSE research into the enterprise software engineering education. *Learning* serves as the bridge that connects research.

The research may be drawn from an ongoing or a completed project, and also from projects led by software engineering professor or by others. The author has integrated her own research with her teaching [2-4][11][12]. She has also incorporated others' research [13][14]. A large, diverse body of students has ranked this effort most beneficial.

Partnering with Industry in Education and Research

Industry has long experienced the lack of adequate preparation of too many software engineering graduates [9][15]. Universities have long experienced the tension between an internal value system that emphasises education in enduring principles and the demands of industry for training in current technologies; neither extreme is appropriate [9][15][16]. The emphasis on the long-term view is what differentiates the partnership in education from a training exercise [9]. Indeed, the goal of industry-academic partnership needs gain the best of both worlds: industrial involvement and advice and academia's long-term view of what makes a quality engineering education [9].

Integrating Diversity into the Programme

It is crucial to the vitality of engineering that students learn how to be a productive member of a diverse workforce. Such learning opportunities can be provided through forming diverse teams to work on various team-based projects and assignments.

CONCLUDING REMARKS

As software becomes ubiquitous and plays an ever increasingly important role in systems of great significance, making software engineering education flexible and responsive to technological changes becomes more critical. The enduring principles and models at the core of the software engineering curricula change more slowly than the examples of current practise; however, they change more rapidly than the core of other fields. Changes in software technologies and software development models require proportionate changes in the software engineering education.

Software engineering education programmes are expected to produce an adequate supply of proficient software engineers who are well prepared to develop, deploy, maintain and evolve a wide variety of agent-based and agent-rich enterprise and mission-critical software systems. This requires restructuring software engineering education.

In order to facilitate such restructuring efforts, this article has presented key strategies that are confirmed to be effective in a similar effort led by the author. These strategies are mainly for building a solid foundation for life-long learning, integrating research and education, establishing industry-academic partnership in research and education, and integrating diversity into the programme.

REFERENCES

1. Griss, M. and Pour, G., Accelerating development with agent components. *IEEE Computer*, 34, **5**, 37-43 (2001).
2. Pour, G., *Web-Based Multi-Agent Architecture for Software Development Formal Peer Inspection*. In: Mohammadian, M. (Ed.), Computational Intelligence for Modelling, Control, and Automation Series. Burke: IOS Press (2003).
3. Pour, G. and Guo, Y., A Web service-based architecture for supply chain management. *Internet Computing Series* (2003).
4. Pour, G. and Hong, A., *Internet-Based Multi-Agent Framework for Software Service Retrieval and Delivery*. In: Mohammadian, M. (Ed.), Computational Intelligence for Modelling and Control Series. Burke: IOS Press (2003).
5. Pour, G., Web-based architecture for component-based application generators. *Internet Computing Series*, 403-409 (2002).
6. Pour, G., *Component Technologies: Expanding the Possibilities for Component-Based Development of Web-Based Enterprise Applications*. In: Furht, B. (Ed.), Handbook of Internet Computing. Boca Raton: CRC Press (2000).
7. Pour, G., Component-based software development: new challenges and opportunities. *TOOLS Series*, IEEE Computer Society Press, 26, **8**, 376-383 (1998).
8. Lowry, D.D. and Lowry, M.R., Legal issues in knowledge software engineering. *Proc. 10th Knowledge-Based Software Engineering Conference*, 61-69 (1995).
9. Pour, G., Griss, M. and Lutz, M., The push to make software engineering respectable. *IEEE Computer*, 33, **5**, 35-43 (2000).
10. Fink, L.D., *Creating Significant Learning Experiences* (4th edn). New York: Wiley (2003).
11. Pour, G., An innovative approach to integrating research into education in Component-Based Software Engineering (CBSE). *Proc. 6th UICEE Annual Conf. on Engng. Educ.*, Cairns, Australia, 105-108 (2003).
12. Pour, G., Integrating agent-oriented enterprise software engineering into software engineering curriculum. *Proc. Frontiers in Educ. Conf.*, Boston, USA (2002).
13. Cowan, D. and Griss, M., Making software agent technology available to enterprise applications. *Proc. 1st Inter. Workshop on* Challenges in Open Agent Systems (AAMAS'02), Bologna, Italy (2002)
14. Griss, M., Letsinger, R., Cowan, D., Sayers, C., Van Hilst, M. and Kessler, R., CoolAgent: Intelligent Digital Assistants for Mobile Professionals - Phase 1 Retrospective. HP Laboratories Report HPL-2002-55(R) (2002).
15. Shaw, M., *Software Engineering Education: a Roadmap*. In: Finkelstein (Ed.), The Future of Software Engineering. New York: ACM Press (2000).
16. Pour, G., Component-based development refining the blueprint of software engineering education. *World Trans. on Engng. and Technology Educ.*, 2, **1**, 45-48 (2003).