World Transactions on Engineering and Technology Education
Vol.2, No.1, 2003

Component-based development refining the blueprint of software engineering education

Gilda Pour

San José State University
San José, United States of America

ABSTRACT: An effective software development strategy is critical to an organisation's success in achieving its key business objectives, including the effective use of resources, better time-to-market and adaptation to changes in business needs and requirements. Rapidly rising demand for more flexible, adaptable, extensible and robust complex enterprise software systems cannot be met unless software development makes a transition from a craft activity, involving informal kind of reuse (eg code sharing and design patterns), to a modern industrial process capable of using systematic reuse strategies based on Component-Based Software Development (CBSD) and Agent-Oriented Software Development (AOSD), which extends CBSD. The paper discusses the necessity of refining the blueprint of software engineering education in order to make the transition from traditional software development to CBSD and AOSD. The paper also presents a new practical approach for increasing the effectiveness of the learning experience by integrating CBSD/AOSD research into the software engineering curriculum and providing students with the foundation for life-long learning to help enable them to expand their engineering knowledge and skills throughout their careers.

INTRODUCTION

Critical to an organisation's success is an effective software development strategy that can assist it in accomplishing its key business objectives. This incorporates the effective use of resources, better time-to-market and adaptation to the changes in differing business needs and requirements.

There has been a rapidly rising level of demand for more flexible, adaptable, extensible and robust complex enterprise software systems, which cannot be achieved unless software development makes a transition from a craft activity that involves an informal type of reuse (eg code sharing and design patterns), towards a modern industrial process that is capable of using systematic reuse strategies.

Systematic Reuse Strategy

A systematic reuse strategy emphasises reuse throughout the software development lifecycle, as well as the reuse of highly flexible software components across multiple applications and projects. Traditional software development strategies and engineering methodologies, which require the development of software systems from scratch, fall short in this regard. Component-Based Software Development (CBSD) and Agent-Oriented Software Development (AOSD), which extends CBSD, offer attractive alternatives in order to develop complex enterprise software systems that are flexible, adaptable, extensible and robust.

Component-Based Software Development

CBSD is based on developing and evolving software systems from selected pre-engineered and pre-tested reusable software components, then assembling them within appropriate software architectures. The CBSD approach delivers the promise of large-scale software reuse by promoting the use of software components built by various developers (ie commercial vendors and in-house software developers). CBSD 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.

AOSD extends CBSD; it allows developers to use a set of high-level, flexible abstractions to represent a variety of enterprise software systems. The rapid integration of distributed agents provides opportunities to build enterprise software systems. A software agent component (also known as an agent) is a specialised component that exhibits a combination of several of the following characteristics:

- Autonomous;
- Adaptable;
- Mobile;
- Knowledgeable;
- Collaborative;
- Persistent.

Furthermore, agents offer greater flexibility and adaptability than traditional components [1].

Making a Transition to Component-Based Development

The availability and growing popularity of component technologies, such as JavaBeans, Enterprise JavaBeans (EJB) and .NET, have helped facilitate the transition from traditional software development to CBSD. However, using component technologies and methodologies is not sufficient for establishing component-based software engineering business.

Making a transition from traditional software development to CBSD/AOSD requires explicitly defining software reuse in software lifecycle processes in order to allow an organisation to exploit, systematically and repeatedly, reuse opportunities in multiple software projects and software products. Without such a repetition, any improvement to the software lifecycle and software products, which result from large-scale software reuse based upon CBSD/AOSD, will be limited and often disappointing [2-4].

INTEGRATING COMPONENT-BASED DEVELOPMENT INTO SOFTWARE ENGINEERING CURRICULUM

In order to prepare software engineering graduates to practice component-based and agent-oriented software engineering, it is necessary to integrate CBSD and AOSD into the software engineering curriculum. This would incorporate emphasis on the differences between the traditional software development lifecycle and the CBSD/AOSD lifecycle, new software engineering roles requiring new competences and skills, technical and non-technical issues facing CBSD/AOSD, as well as ways to address these issues [5][6].

New Roles Requiring New Competencies and Skills

The CBSD/AOSD lifecycle differs from the traditional software development lifecycle in many ways. For instance, the following elements are part of CBSD:

- The design phase involves selecting and customising enterprise software architectures and a set of software components. It also requires system developers to architect and design extensibility and scalability into the enterprise system and all its subsystems.
- The implementation phase involves building software, such as wrappers and mediators, to support the required interactions among various software components within appropriate software architectures.
- The testing phase involves addressing issues raised by the late integration of components that have been developed by others, and the lack of confidence in, and understanding of, components built by others.
- The maintenance and evolution phase involves addressing issues raised by the lack of confidence in, and understanding of, components built by others, etc.

Furthermore, three main categories of processes have been identified in CBSD/AOSD, as follows:

- The component system engineering process staffed by a team for each component system.
- The application system engineering process staffed by a team for each application system.
- The application family engineering process staffed by a team for the entire layered system, which is particularly focused on defining the division into systems and the interfaces between them in order to support system interoperability [2][3].

Due to the above-referenced key differences between the traditional software development approach and CBSD/AOSD, different sets of competences and skills are required of software engineers when working in different phases of the software development lifecycle and different categories of CBSD/AOSD processes.

It is essential to provide educational and training opportunities for software engineering students and professionals to understand the differences between the traditional software development approach and CBSD/AOSD, and to acquire new competences and skills that are required for practising CBSD and AOSD. Table 1 shows a list of different competency units in CBSD and the related types of workers.

Table 1: Competency units and worker types for CBSD.

Competency Units	Worker Types
Requirements Capture Unit	• Use case engineer • GUI coordinator • GUI engineer
Design Unit	• Subsystem engineer • Use case designer • Design subsystem engineer
Testing Unit	• Tester • Test engineer
Component Engineering Unit	• Reuse process engineer • Reuse support environment engineer • Façade engineer
Architecture Unit	• Architect • Lead architect • Distribution engineer
Component Support Unit	• Component system trainer • Component system supporter • Component system librarian

The learning opportunity can be provided through a team-based term project and assignments that emphasise both the theoretical and practical aspects of the course topics in any one of the courses in the new course sequence on CBSD and AOSD.

It is critically important to build a foundation for life-long learning to help enable software engineering students to enhance their engineering competency and skills throughout their careers. It is also essential to highlight the relationship between software engineering and enterprise applications [7]. Further, other factors that need to be accentuated include: a broader understanding of information technology, systematic thinking, cooperative problem solving, the ability to communicate effectively, a systems view of problem solving and technical and economic decision making that is required in enterprise software development.

Integrating Research into Software Engineering Education

To stimulate innovative educational activities in the software engineering discipline, and to increase the effectiveness of students' learning experiences, it is essential to integrate CBSD/AOSD research into software engineering courses and curricula.

The research may be ongoing or completed, and may be drawn from any research project in the software engineering field. The author has integrated the results of her research into the new CBSD/AOSD course sequence that she has developed [8-11]. This work has been accomplished as a part of her multi-year project, which has focused on integrating CBSD and AOSD into the software engineering curriculum.

In addition, the author has integrated the results of comprehensive experimental research carried out by Ivar Jacobson and Martin Griss, two distinguished experts in modern software engineering, into a new course on component-based software engineering.

Jacobson and Griss have been deeply involved with major reuse efforts at Ericsson and Hewlett-Packard Company, and have closely examined the experience of a number of other large organisations with software reuse programmes [2]. Jacobson and Griss have reported several technical and non-technical issues that block or hinder the transition from traditional software development to CBSD [2]. The issues are categorised as follows:

- Business-oriented issues that are due to the lack of adequate funding for initial investment, education and training, and access to vendor-supplied components; the lack of convincing business case and economic model for long term investment, and clear definition of product-line mode and features.
- Process-related issues that are due to low process maturity of the organisation, ill-defined or unfamiliar reuse-oriented methods and processes, new inter-group coordination and management needs, well tested and documented methods and models to relate features to component sets and variability, etc.
- Organisational issues that are due to the lack of a systematic practice for reuse activities and enterprise component development, lack of management expertise and support, etc.
- Engineering issues that are mainly due to the lack of adequate techniques and tools to identify, design, document, test, package and categorise software components; the lack of adequate and well understood standard patterns and architectures, etc.
- Infrastructure-related issues that are due to the lack of widespread use of common tools, base components, standardised design notation, such as the UML, different programming languages and environments, support for multi-group configuration management, etc.

To address the above-referenced issues raised in the transition from traditional software development approach to enterprise-wide reuse business based upon CBSD, Jacobson and Griss developed a systematic and incremental transition process to CBSD [2]. Their experimental research has concluded that it is critically important to obtain the support of both management and software development teams for the systematic and incremental transition process.

It is essential to involve customers, users and maintainers at an early stage in the systematic transition process. The emphasis should be on component-based infrastructure and architecture [12]. Other factors include a strong level of commitment from top management in order to provide the required support and investment for the practice of developing families of complex software systems from well-designed and architected reusable software components, organisational support for reuse efforts, a stable application domain, standards; as well as the required level of education and training.

Major elements of the transition process are as follows:

- An assessment of business, process, domain and organisational readiness for component-based software engineering.
- The design of a multi-step, pilot-driven transition plan to install the new process into an existing software engineering organisation.
- The customisation of the generic component-based engineering enterprise and organisation design.
- The use of well-designed architectures for family applications in order to support the customisation required for the development of component-based enterprise software systems.
- Testing, tool development and deployment.

This systematic and incremental transition process has been put into practice at Hewlett-Packard Company and Ericsson for the past several years. It has produced incredible results, including major development time and cost savings and product quality improvements [2].

Moreover, other industries have all reported major benefits including significant cost and time savings from a systematic and incremental transition process to CBSD. These include the following:

- The aerospace industry (eg Boeing and Lockheed);
- Computer companies (eg Hewlett-Packard, IBM, Microsoft and Toshiba);
- Banking (eg Citicorp, Chemical Bank, First Boston, Montreal Trust and Mellon Bank);
- Government bodies (the Army, Air Force, NASA and Navy);
- Insurance companies (eg Hartford, Northwest Mutual Life and USAA);
- Manufacturing enterprises (Hewlett-Packard, Foxboro and NEC);
- The telecommunication industry (eg AT&T, BNR, Ellemtel-Ericsson, GTE, Motorola, NEC, NTT, PacBel, Sprint and MCI);
- Utilities (eg NEC and Sema-Metra) [2][11].

In almost all cases of successful enterprise reuse, the keys have been management support, system and component architecture, a dedicated component group, stable application domain standards and organisational support.

Industrial-Academic Partnership in Education and Research

A deep, sustained partnership encourages the development of more effective software engineering education programmes and ensures that university research will have greater access to, and influence on, industrial-scale development. The goal is to get the best of both worlds: industrial involvement and advice and academia's long-term view of what makes up quality education. This emphasis on the long-term view is what differentiates the partnership in education from a training exercise [13].

Because many large companies have had to mount significant software engineering training programmes, mainly due to the dearth of software engineering education at the college level, they should be more than willing to assist in nurturing software engineering programmes that emphasise the development of core competences and skills required for CBSD and AOSD.

Effective industry involvement is not trivial; goals and investments must match. Typically, academia wants to emphasise fundamentals and concepts, as well as life-long learning, while industry focuses on acquiring skills in order to fill an immediate need. The key is to achieve the right balance and there is often more than one way to achieve that. However, it is important that institutions provide opportunities where individuals may concurrently assume responsibilities as researchers, educators and students; and where all can engage in joint efforts to enrich both research and education.

The partnership should focus on helping universities arrive at the appropriate balance between fundamental knowledge and its engineering applications [13].

Integrating Diversity into Term Projects

It is important to provide opportunities for software engineering students to learn how to work and contribute effectively within a diverse team of software engineers and IT professionals. This learning opportunity can be provided through term projects and class activities.

CONCLUDING REMARKS

In order to meet the ever-increasing demand for more flexible, adaptable, extensible and robust complex enterprise software systems, software development has to make a transition from a craft activity involving an informal kind of reuse to a modern industrial process that is capable of using systematic reuse strategies based on Component-Based Software Development (CBSD) and Agent-Oriented Software Development (AOSD).

A key element in making the transition to CBSD/AOSD is refining software engineering education so as to ensure software engineering students and professionals acquire the knowledge, competences and skills that are required for practicing CBSD and AOSD.

It is critically important to the success of software engineering education and the profession to integrate the results of related research as well as the fundamentals and key concepts of CBSD into the software engineering curriculum [14]. It is also essential to ensure that graduates of software engineering programmes have the necessary foundation for life-long learning so as to help enable them to expand their engineering knowledge, competences and skills throughout their careers. It is also essential to provide opportunities for software engineering students to practice and learn how to work and communicate effectively within a diverse group.

To refine software engineering education and advance the software engineering discipline, industry-academia partnerships in education and research play a major role. The focus of the partnership should be on helping universities develop and implement educational and training programmes with the appropriate balance between fundamental knowledge and its engineering applications. To do so, institutions should provide opportunities for researchers, educators and students to engage in joint projects, internships and fellowships in order to enrich both research and education.

REFERENCES

1. Griss, M. and Pour, G., Accelerating development with agent components. *IEEE Computer*, 34, **5**, 37-43 (2001).
2. Jacobson, I., Griss, M. and Jonsson, P., *Software Reuse: Architecture, Process and Organization for Business Success*. Reading: Addison Wesley (1997).
3. Pour, G., Griss, M. and Favaro, J., Making the transition to component-based enterprise software development: overcoming the obstacles - patterns for success. *TOOLS*, 29, **6**, 419-420 (1999).
4. Pour, G., Component-based software development: new challenges and opportunities. *TOOLS*, 26, **8**, 376-383 (1998).
5. 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).
6. Pour, G., Integrating agent-oriented enterprise software engineering into software engineering curriculum. *Proc. Frontiers in Educ. Conf.*, Boston, USA (2002).
7. Murch, R. and Johnson, T., *Intelligent Software Agents*. Englewood Cliffs: Prentice Hall (1999).
8. Pour, G., Web-Based Architecture for Component-Based Application Generators. *Internet Computing Series*, 403-409 (2002).
9. 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).
10. 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).
11. 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).
12. Lim, W., *Managing Software Reuse*. Englewood Cliffs: Prentice Hall (1998).
13. Pour, G., Griss, M. and Lutz, M., The push to make software engineering respectable. *IEEE Computer*, 33, **5**, 35-43 (2000).
14. Brown, A., *Large-Scale Component-Based Development*. Englewood Cliffs: Prentice Hall (2000).