Mohamed E. Fayad and Wei-Tek Tsai

The stakes are increasing every day as more companies bet on object-oriented technology for military and commercial application development. Instead of being an academic exercise, making a commitment to use object-oriented technology is a very serious managerial and technical decision—one which affects careers, projects, and entire organizations.

On the positive side is the promise of higher quality and more competitive applications when used effectively [2]. Object-oriented technology (OOT) promotes a better understanding of requirements and results in more modifiable and maintainable applications, providing other benefits such as reusability, extensibility, robustness, reliability, and scalability. OOT promotes better teamwork, good communication among team members, and a way to engineer reliable software systems and applications.

Authors in this special section describe knowledge gained from applying object-oriented techniques, tools, and environments to a variety of software development projects.
Some Lessons Learned in Transitioning to OO Software Development

These lessons provide good tips for smoothly completing the transition to object-oriented software engineering (OOSE). Some of these lessons are presented in more detail in [2].

Lesson 1: The transition to OOSE is a mission with problems. The adoption of a new paradigm creates fear and uncertainty: fear of “getting it all wrong” and uncertainty about the process to use. Organizations often begin OOSE projects without any software engineering process to use as a foundation. In addition, OOSE complicates project manager roles in training, staffing, standards, documentation, scheduling, planning, cost estimation, data collection and software metrics.

Lesson 2: There is little guidance for OO software development project managers on how to transition to OOSE. Project managers must find their own way through the maze of techniques, tools, and environments with little or no guidance from published sources. Many OOSE training activities provide information biased toward a particular technique. OOSE complicates the project manager’s role and creates several problems related to planning, staffing, training, scheduling, cost estimation, documentation, legacy systems, and software metrics.

Lesson 3: Careful selection of the first object-oriented project is very important. To adapt the OO technology well for the first time, the first project must possess three major characteristics:

1. It must be a new project. From our experience, the new project offers the best possibilities. If an existing project is selected, a large number of issues must be dealt with, such as legacy systems, old and established development processes, and existing staff. A new project often has sufficient “up front” time to allow for training and experimental prototyping.
2. It must be large and meaningful enough to influence the other project members’ attitudes toward the OO technology.
3. It must be supported by the customer and/or high-level management. Initiatives die without customer or management support.

Lesson 4: Careful pre-project planning will smooth the transition from a chaotic software development process.

Lesson 5: The software development plan (SDP) is one of the most critical documents: it plays an important role in any software development project and must be the first document produced. A small business must have a business plan before talking to the bank, the small business administration, and investors. We often see millions of dollars (sometimes billions) allocated for software development projects with no planning. The SDP:

• Is the controlling document for managing a software development project.
• Provides a definition for each major task, an estimate of the time and resources required.
• Provides a framework for management review and control. In almost every situation in the U.S. Department of Defense (DoD) environment, a draft of an SDP must be provided by the software development contractor and included as part of a prospective contractor’s proposal.

Lesson 6: Technique selection will impact virtually every activity in the new software development process.

Lesson 7: Extensive testing and verification and validation (V&V) during the development of software is essential for building zero-defect software [1]. Unfortunately, most V&V activities are ad-hoc. Verifying and validating OO specifications and programs are still very young research topics and more research is needed. Recent attempts include multilevel specification checking [4] and test-case generation from methods and message specifications [4]. Kirani [3] points out that if successful testing and V&V activities for the OO paradigm are not developed, there is a great risk of failure of the OO paradigm as a next-generation software development technique.

Lesson 8: Staffing the first OO project requires special consideration. First-time OO project managers will find that their previous organization may not be well-suited to OO activities. There may be a need to define or change the definition of some team roles. The people selected for this team should be eager to learn new concepts.

Lesson 9: Development teams or developers’ attitudes and attributes are one of the major keys to success or failure of adapting the OO technology on the project. Individuals must be selected carefully and possess two personality traits: aggressiveness to push forward new ideas and receptiveness to gain knowledge from a service-oriented demeanor.

Lesson 10: Training should be spread over a number of weeks. A requirement for 40 hours of training may be filled in one week if that week is totally dedicated to the training activity. The same training may be spread across two weeks if training is only half-time. The training may also be spread across 10 weeks with the addition of a part-time consultant. We favor the later approach for several reasons. Compressing the training tends to overwhelm the students—they are exposed to too much, too fast. Spreading the training across a longer period allows the student time to reflect on and practice each significant detail.

Lesson 11: Software development processes map the abstract theories of the OO technique into concrete and repeatable actions.

Lesson 12: A well-defined process is essential. Our experience in applying a new OO technique emphasizes the importance of having a well-defined software process. Software development processes ensure the software products have a consistently high quality, and using an unfamiliar development technique makes them even more important. Adapting a new development technique will undoubtedly change the way developers do their jobs, and a document development process helps them understand what is expected of them during each development phase. It is also important that team members are enthusiastic about the new development technique—the manager must ensure they understand the development technique and how it has been adapted to their specific project.

Lesson 13: Metrics provide a means of measuring process quality and identifying potential bottlenecks.
The use of OOT is beneficial in creating software applications that satisfy human needs and meet the following objectives:

- Operational or functional objectives, such as reliability and efficiency.
- Development of nonfunctional objectives, such as understandability, reusability, and maintainability.
- Managerial objectives, such as better teamwork, productivity, and engineering the right products.
- Business or economical objectives, such as cost efficiency leading to higher return on investment.

OOT offers the promise of applications that can be quickly extended to satisfy the changing requirements of customers (or increase the useful lifetime of the application). It promotes applications that are shielded from the continual flux in operating systems and programming language technologies. All of this is the magic of OOT: one change in concept that leads to a multitude of benefits that yield higher-quality software products.

On the negative side is the significant learning curve involved in bringing teams of system software project organizations, including requirements analysts, system engineers, software engineers, software designers, and software managers, up to a level of competence on OOT [2]. This learning curve implies a longer initial time to market or a longer development time for initial projects, which is often a bitter pill to swallow. In addition to introducing a new way of developing software, object-oriented technology may require new tools, new programming languages, new metrics, and new software development processes.

Transitioning to object-oriented software engineering (OOSE) is a task with many potential surprises. For example, transitioning to OOSE complicates the software manager’s job. This transition requires the manager to deal with a new or different set of problems: staffing, training, scheduling, planning, object-oriented processes, tools, cost estimation, standards, documentation, metrics, and transition process. Most of these are mostly old problems, just realized in a new context. While managers may already address some of these areas, OOSE requires a modified approach to this management task. The authors of articles in this special section have encountered many of the surprising aspects of OOT, will introduce you to them, and help you become familiar with these situations so they are no longer surprises.

The future of object-oriented technology will bring us a mixture of satisfaction and disappointment. We should expect the disappointment because developing software is typically a painful process, especially when new ground is being broken by the use of new processes, such as techniques and new developments. The lessons learned from the satisfying experiences will help promote the technology inside the organization in the long term. You should seek to understand the benefits of OOT and learn how to utilize it to your organization’s advantage. Alleviating the problems of OOT will lead to higher-quality software products, such as class libraries and application frameworks, that provide cross-platform portability. Tools will evolve that maximize the productivity of software personnel and minimize process downtime or time lost from problems with the development process that cause developers to be unsure of how to proceed. New methods and techniques will emerge that help software development teams over the object-oriented technology hurdle and reduce the learning curve. Software products will help people manage complex projects more efficiently. The situations and experiences presented here will benefit the work of subsequent efforts.

The object-oriented experiences explored in this section focus on lessons learned and appreciation gained from transitioning and applying object-oriented technology. These lessons are presented from two distinct perspectives: managerial (see accompanying sidebar and article by Berg, Cline and Girou) and technical. The article by William Berg, Marshall Cline, and Mike Girou describes the lessons learned during the transition of a team of 150 developers to OOT. Douglas Schmidt discusses his experiences applying a design pattern-based reuse strategy to develop OO software frameworks for several distributed systems in Ericsson and Motorola’s IRIDIUM project and Kodak’s Health Imaging Systems. David Kung et al. analyze the development of an OO testing and maintenance environment. Charles Norton et al. describe their experiences with Fortran 90 and C++ in developing OO parallel computation for plasma PIC simulation.

Acknowledgments
We would like to thank Milton Fulghum for his comments on this introduction. We would like to thank all the anonymous reviewers for helping us selecting the best articles for this section.

References

Mohamed E. Fayad is an associate professor of computer science at the University of Nevada, Reno. Wei-Teh Tsai is a professor of computer science at the University of Minnesota.