What Does Industry Require?

Organizations using object technology (OT) for development of software products need to be confident that their way of working is likely to produce a good quality product — on time and within budget. Some framework is needed within which technical details of the analysis, design, and coding can proceed and within which managerial control of the project can be exerted. That framework is an OO lifecycle methodology. This methodology should cover the whole lifecycle, should embody a general underlying philosophy regarding the degree of iteration necessary when using OT, and should provide management and technical techniques. It should, \textit{inter alia}, delineate deliverables (type and timing), offer support for a hierarchy of abstraction levels (e.g., class, subsystem, pattern), ideally embody the notions of contracting (e.g., [60]), offer an overlay of complexity management techniques (e.g., layering, sheets, subsystems, selective visibility), and provide metrics for evaluating quality aspects of both product and process. Management techniques will focus on deliverables and quality; technical techniques will provide a means of creating the design and translating that into code. An OO methodology also needs to supply assistance with library management for creating new reusable classes and for using existing reusable classes. The increasing sophistication of software development environments
also means that most methodologies should have identifiable tool support, such as a drawing tool to support the notation embodied within the methodology.

A methodology provides an organization standard — not rigid but more of a skeleton within which both the engineering and the craft aspects of software development may be supported. It provides guidance and support but is not a cookbook to be followed blindly. But it does provide a sufficient level of standardization to form the basis for optimum project management of the software development process [86]. Some methodologies, such as SOMA, MOSES and OPEN, provide clear lists of deliverables. Whilst retaining an iterative development process, such methodologies also offer the capability of associating these deliverables with progressive payments when a third party developer providing tendered contracts is used [85]. Without this methodology-provided framework, payments are only possible in an ad hoc fashion — thus demeaning this external contract to a low level on the SEI’s Capability Maturity Model (CMM).

Despite small differences between methodologies, there is clearly much common ground. Whilst “borrowing” between methodologists continues — inevitably leading to concurrence of process, notation, and techniques — there is also interest in creating standards.

At a pragmatic level, many project leaders and senior managers prefer to adopt, if not a (de facto) standard, at least an approach in reasonable use. This ensures that the chosen methodology is supported by book(s), training, and availability of CASE tools. Perhaps most importantly, there is a pool of software engineers skilled in this approach. (Of course, the corollary to being able to acquire new employees from this pool is that you are more likely to lose your best staff to competition using this same methodology).

---

1 It should of course be noted that a cookbook is only viewed as a strict ordering of operations by a novice cook. Experienced cooks merely regard a cookbook as a point of departure for their creativity, e.g., [78, p. 77]. In that latter sense, a methodology should be a cookbook.