Mohamed E. Fayad, Mauri Laitinen, and Robert P. Ward

Software Engineering in the Small

Smaller-sized software companies are developing significant products that need effective, tailored software engineering practices.

In 1968, the NATO Software Engineering Conference in Garmisch, Germany [6] initiated the concept of software engineering, identifying the problems with producing large, high-quality software applications. In 1975, De Remer [2] introduced the terms, "programming in the small" and "programming in the large" to differentiate the development characteristics of large-scale software development from detailed programming (for example, data structures and algorithms). The principal source of large-scale software at the time was development contracts issued by the U.S. Department of Defense. Since then, virtually all software engineering literature has concentrated explicitly and implicitly on the model of DoD contract software development.

Since the late 1970s, the microcomputer revolution has dramatically increased the quantity of software produced, the average size of programs, and the number of companies involved in software development. Much more software is produced for internal use, commercial applications, and the mass-market than for deep-pocketed government and large industry. Using the number of units sold, mass-market software dwarfs the other forms of software sales.

The growth of the software industry has produced many small companies that do not do contract software, but rather compete in other areas. This gives rise to at least four significant development issues that have not been adequately addressed in software engineering literature: company size, development mode, development size, and development speed. We discuss these issues and then discuss some of the shortcomings of current software engineering thinking for small companies.

Company Size

Definitions of "small" businesses vary by industry and by government agency from 100 to 500 employees or more. These bounds are somewhat broad for our purposes. Based on census data, we define companies of 50 or fewer employees as small. According to the U.S. Census Bureau's "1995 County Business Patterns," using Standard Industrial Codes (SIC) to categorize business types, the vast majority of software and data processing companies fit our designation. The data in the Table shows that for companies whose primary function is developing software, those with more than 50 employees comprise only a few percent of the total. SIC categories 7371 and 7372 denote computer programming services and prepackaged software respectively. These two subcategories make up 45% of the total industry.

We suspect that these figures are conservative. The American Electronics Association (AEA) [1] reports the rate of small startups has continued to grow and

the prepackaged software segment of the industry has grown dramatically. Moreover, these numbers do not include companies whose primary product is not software but have major software investments. While large companies have a greater total employment, every small software company must have at least one person versed in software development.

Development Mode

Much software engineering literature implicitly assumes the contract model for software development in the sense of identifying a customer for whom the work is being done. The usual approach for noncontract software development is to name some internal department as the customer. But this doesn't always work well. For enterprise-wide software, the customer may be every department in the company, not one well-focused group. In addition, internal development rarely uses a firm written contract to develop software where all parties freely agree to the terms and conditions.

Organizations that develop prepackaged software, especially small companies, generally do not use the contract model. For small companies, survival rests on finding actual, paying customers. The idea that prepackaged software is being developed for an internal customer, usually marketing or product management, does not accurately reflect the way small companies think about product development. Using an internal surrogate "customer" requires resources and time a small company cannot afford, and it can divert the focus from satisfying a market need. It also downplays the extensive involvement software development people have in product specification.

Development Speed

While hype abounds about faster development, many software markets have become fiercely competitive and have demanded faster software product delivery. As a result, some new rapid development strategies have been developed [3–5]. Among the new approaches are phased development and delivery, overlapping design, development, and testing, and use of third-party components and application frameworks. These strategies have significant impact on software engineering practices

Development Size

While a 100,000 source line program was a significant undertaking 20 years ago, the typical shrinkwrapped software product today embodies at least that many lines of code. While it is extremely difficult to identify a cost figure, it appears that smaller groups are developing larger programs. This suggests that smaller groups need some of the software methodologies developed for large-scale projects, but they don't need all of them. A major problem with the large-scale methodologies is they do not scale down well.

While contract software is no less important than it was, there are many more types and modes of development that deserve serious investigation from a software engineering standpoint. Mass-market software companies have their own set of development and maintenance issues relating not only to their size but to dealing with the large volume of shipped products. For example, specialized forms of development and testing are needed for the effective distribution of maintenance and feature updates. Startup companies have development needs that are almost in a class by themselves. As we discuss, development speed is a primary concern while development for reuse may be consciously avoided. These specialized needs are combined with the challenges of framework and component integration.

How Software Engineering Overlooks Small Groups

Looking at the software engineering literature, it is surprising that the organization type and size play so small a role in discussions of the development process. For example, according to the American Electronics Association, startup companies play a significant part in the booming software economy, but literature discussing the issues of startups in terms of software engineering methods or economics is virtually nonexistent. Startups are typically small companies where resources are limited, competition is fierce, and time to market is a primary driver of development. These characteristics have a number of implications for traditional software engineering concepts:

- Developing for reuse, a prominent issue in software engineering, may not be an advantage. The added cost, and especially the increased time needed to assure reusability, are less impor-

tant than releasing the product. Note that reusability will be important if the company is successful. This brings up some interesting issues. Engineering for reusability in subsequent releases may cost more than usual and may require a more extensive redesign. Even so, it may be the only viable way to meet initial release requirements. While this model comes closer to Brooks' "Plan to throw one away" recommendation, little research has been done on this issue.

- There is no historical base for estimating project costs and time. The initial project is the most important, but it has the most variability and the least data to go on. In particular, estimation models based on similar projects are not easy to apply to innovative projects with new teams. Note that this situation is entirely counter to the standard recommendations for effective software estimation. (For example, see [5].)
- Clear and stable requirements are unlikely to exist. First, the company must develop its own requirements rather than relying on even the initial specifications of a customer. Second, competing products can force the organization to make major requirements changes during mid-development.
- Resources are severely limited. Many recommended practices, such as having separate QA and development teams, having component librarians, project managers, and such are not part of the startup's landscape. Many elements of larger project management are not only not needed but could be detrimental.
- Incremental development and release is a increasingly popular mode of development [4, 7]. This suggests that full analysis and design does not occur before the first customer shipment. In extreme cases, "unfinished" or "beta" software is shipped to customers. While we may decry this latter trend, it appears to be what many developers do and what even more customers want.

Table 1. Breakdown of data processing companies by number of employees.

Number of employees	Number of companies	% of companies in this size range	Total number of employees
1- 4	48,773	64.9	65,411
5- 9	9,846	13.1	65.055
10- 19	6,724	8.9	90,857
20- 49	5,392	7.2	164,403
50- 99	2,348	3.1	163,607
100- 249	1,498	2.0	227,155
250-499	387	0.5	131,779
500-999	129	0.2	87,496
> = 1,000	75	0.1	121,7712
Total	75,172	100.00	1,117,475

Examples of Software Engineering Mismatches

The following are a few examples of other areas in which the assumptions made in software engineering literature about the development environment do not match actuality.

Software cost considerations. Cost per source line is not always meaningful in many small organizations. For example, it is usual for startup organizations to care only about the time and resources required to accomplish the goal, and the cost of software is a secondary consideration. Since there is no particular benefit to calculate cost by source line, it is not used. Again, in startups, even though reuse might play an important role if the company is successful, it has far less value than other characteristics, such as time-to-market.

Although direct reuse is often not so important to startup programs, the cost benefits of using commercially available components and frameworks are. It is now customary to use a GUI and development environment framework. When available for a particular segment, components also allow developers to focus on the core development.

Cost over time. It is true that software tends to become costlier over time. There are multiple causes for this, but among those most prominent are increased size and functionality, multiple hardware and operating system platforms, and the requirement for backward compatibility, which increases the analysis and testing costs. In addition, as products become more widely used, the cost and complexity of customer support usually increases and the demand for changes and corrections increases. Problems reported by customers must often be researched by either development or maintenance personnel, even if the end result is that a problem is merely a misunderstanding. Moreover, marketing costs for prepackaged software

increase as the product becomes more popular. Even if per-unit costs decrease, the total cost (and the risk exposure) increase. Note that in many cases, the cost of distributing corrections and updates can be extremely expensive. Also note that for a small prepackaged software company, maintenance costs are not usually borne by the customers and thus can significantly affect the company's profitability. This reality in turn helps dictate the way software is developed. The usual life cycle discussions don't adequately address these areas.

Conclusion

Software engineering in the small is an important area overlooked not only by the literature but by software engineering societies and computing institutes as well. As we point out, the majority of software development organizations are small, but they are developing significant products that need effective software engineering practices tailored to their size and type of business. Furthermore, we assert that existing practices developed for large, contract-based systems do not serve the small organization adequately. In the next "Thinking Objectively" column we will discuss the difficulties of scaling down. C

References

1. American Electronics Assiociation. *Cybernation: The Importance of the High-Technology Industry to the American Economy*, 1997.
2. DeRemer, F and Kron, H. Programming in the large versus programming in the small. *SIGPLAN Not. 10*, 6 (Apr. 1975).
3. Fayad, M. and Laitinen, M. *Transition to Object-Oriented Software Development*. John Wiley & Sons, NY, 1998.
4. Marco, I. and MacCormack, A. Developing products on Internet time. *Harvard Bus. Rev. 75*, 5 (Sept.–Oct. 1997).
5. McConnell, S. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, Redmond, Wa., 1996.
6. Naur, P. and Randell, B., Eds. *Conference on Software Engineering*. NATO Scientific Affairs Division, Garmsich, Germany, 1969.
7. Redmill, F. *Software Projects: Evolutionary Vs. Big-Bang Delivery*. John Wiley & Sons, NY, 1997.

Mohamed Fayad (fayad@cse.unl.edu) is J.D. Edwards Professor at the University of Nebraska, Lincoln.

Mauri Laitinen (mdl@sierra.net) is a principal in Laitinen Consulting at Lake Tahoe, Calif.

Robert P. Ward (robert.ward@moai.com) is a program manager for Maoi, a provider of Internet commerce solutions.