As a sign of its increasing maturity, the software engineering community has begun to accommodate the needs of different development organizations. Organization size, customer needs, product cost, time to market, and product safety all affect the way software will be built. If it is a part of a line of products, compatibility with other products becomes a primary design issue. As our community has become more comfortable with the methods and processes required to build software, we recognize that the methods developed for large systems are not optimal for all systems. In particular, development done by small groups does not always fit into the development model originated to cope with large, contract-based software.

Software engineering originally targeted large, contract-based development for defense and industry, and the discipline has achieved significant improvement in that area. Although contract software is no less important than it was, there are so many more types and modes of development that deserve serious investigation from a software engineering standpoint.

There are three reasons to advance the study of software engineering in the small:

- changes in the industry,
- underestimates of the number of small companies, and
- the differences in small group development methods from the standard models.

Industry Change

Although the predicted Y2K cataclysm fizzled, even those people who hitherto hadn’t paid much attention became acutely aware of how much the industrialized world relies on computers and on the software that runs them. That growing awareness of our dependence on computers forces us to recognize the importance of engineering reliable software.
The six articles in this section are evenly divided between European and US contributors and show a particular breadth of application.

The first article, Tore Dyba’s “Improvisation in Small Software Organizations,” examines the idea of structured improvisation in software using a jazz metaphor. Dyba discusses the limitations of rationalistic, best-practice–based approaches applied to small groups in turbulent environments. The article suggests that improvisation leads to exploration of new and better-adapted development methods. Such exploration can also lead to failures. In contrast to failures in the large, small-group failures should be seen as a way of developing new competencies.

In “Applying Product Line Concepts in Small and Medium-Sized Companies,” Peter Knauber, Dirk Muthig, Klaus Schmid, and Tanya Widen present the Pulse software engineering method, which addresses the challenges of small companies that must grow single products into product lines. The article introduces a set of processes to help tie the architecture to product variants that meet the needs of various market niches. The authors discuss the method’s analysis of product scope, domains, common components, customization, and reuse to achieve lines of products from a single core.

“A Software Development Process for Small Projects,” by Melissa Russ and John McGregor, addresses the issues adapting an iterative development process to small groups. Particularly interesting is the idea that processes must be matched to the group to be useful and used. It discusses ways of reducing the number of processes to match group bandwidth while retaining effective project control. As we have discussed elsewhere, the consequences of overloading processes or using ill-adapted ones is magnified in small groups.

The fourth article in this section, Vaclav Rajlich’s “Incremental Re-documentation Using the Web,” introduces a Web-based method of...

The last decade has changed both the character of the software industry and the composition of the companies engaged in development. Mass market, the Internet, and game software are large industry segments, subject to intense competition. With PCs doing the complex computing tasks that once required mainframes, small companies have sprung up to create all sorts of software. Groups of six or fewer software developers now commonly create products in excess of 100,000 source lines in 18 to 24 months.

Mass-market software companies have their own set of development and maintenance issues that relate not only to their size but also to dealing with the large volume of shipped products. For example, effective distribution of maintenance and feature updates requires specialized forms of development and testing. Startup companies have development needs that are almost in a class by themselves. Development speed is a primary concern, even as development for reuse might be consciously avoided. These specialized needs combine with the challenges of framework and component integration that play an increasing role in modern software building. Traditional software engineering has paid little attention to these industry segments that small companies often perform.

Underestimating the Number of Small Companies

The overwhelming majority of companies developing software are small—a fact that’s often overlooked. According to the US Bureau of the Census 1995 County Business Patterns, almost 90% of software and data-processing companies have fewer than 50 employees. The rate of small startups continues to grow, with the industry’s packaged software segment growing especially dramatically.

We suspect that these figures are conservative. According to the American Electronics Association, the growth in software employment has increased dramatically in the last few years. Because industry statistics often omit temporary workers, the growth in software-related work is greater than officially reported. Moreover, these numbers do not include companies whose primary product is not software, but who still develop software for in-house use or for sale. We expect that the 2000 census data will show an even larger shift toward small companies.

Here’s the critical point: at least one person in each company should be versed in appropriate software engineering techniques. Given the preponderance of small and very small companies, we find it all the more compelling that we pay attention to their development needs.

One Size Does Not Fit All

Small groups, especially those developing mass-market and Internet software products, rarely work according to the published formal contract models. Although they are intensely interested in best practices, software engineers in small organizations find it difficult to separate the methods useful to them from the methods needed only to coordinate multiple large groups.

These differences surface in three main ways:

- Management is so richly structured in large organizations that it is difficult to...
Adapt to the small. The specialized functions, reporting arrangements, and metrics are hard to scale down.

Software architecture reflects organization architecture. In software engineering circles, this is known Conway’s law, which states, “Organizations which design systems are constrained to produce systems which are copies of the communication structures of these organizations.”

Many methods such as inspection and testing were created to work on larger, nondistributed groups. These methods need serious reworking to adapt them to smaller groups and groups where the members are not physically colocated.

Although software professionals have known for years that large-scale systems needed the discipline of software engineering to build software that met time, cost, and performance goals, they have only recently started looking at the particular needs of different types of development. We now realize that small-scale software engineering is not just a degenerate case of large-scale software engineering but an important subfield in its own right. We trust that the articles presented here will spur further research work in the field (see the “Articles” sidebar).

References

