Playing to the strengths of small organizations

Jorge Aranda

University of Toronto,
Toronto, ON, Canada
jaranda@cs.toronto.edu

Abstract. By virtue of their size, small organizations can take advantage of many opportunities to develop software efficiently and successfully, and they waste them if they try to emulate their larger counterparts. As software researchers, we should study how small organizations can best exploit those opportunities instead of prescribing solutions that were designed for organizations of a very different nature.

"Today, we suffer from an almost universal idolatry of gigantism. It is therefore necessary to insist on the virtues of smallness, where this applies." —E. F. Schumacher [16]

1 Introduction

Small organizations form a large part of the software industry,[1] but our research community has mostly overlooked their needs and characteristics. This is a serious omission: small software organizations have a number of strengths that are absent in larger organizations, strengths that help them develop software efficiently and successfully and that are neglected if the organization applies processes and practices that were not designed to exploit them.

In this paper I argue that small organizations should not attempt to emulate the processes and practices of larger organizations, but should rather take advantage of the strengths enabled by their size. Similarly, I argue that requirements researchers should recognize these strengths, and design and evaluate techniques that make the best use of them.

2 Small is beautiful

We have known since the earliest days of our field that large software organizations suffer from problems caused by their size. They incur in significant coordination overheads [6], and tend to release products that are less satisfactory than those built by smaller organizations [9]. But somehow many in the

[1] In the United States in 2002, 95% of software development firms had less than 50 employees. They generated 21% of the total income and employed 28% of all employees in the area [7].

software industry assume that the goal of a small firm should be growth, that size is a valid measure of success. To be sure, a large size brings certain benefits: the appearance of stability, the ability to engage in greater and more ambitious projects, the appeal of commanding the work of a large number of employees. And yet there are many rewards for small organizations, rewards that often go unnoticed and unclaimed in their push to become large by behaving as if they were already large. Some of these rewards are psychological and even ethical, such as the joy of working in closely-knit groups and a greater agency over one's own work. That kind of reward may be significant enough to justify a preference for small groups, but it is not the topic of this paper. Rather, I claim that small organizations also have important advantages purely from the point of view of developing software efficiently and successfully. Some of these advantages relate specifically to their requirements elicitation and communication activities.

2.1 Formality is unnecessary

Organizational scientists tell us that increases in organizational size lead to greater bureaucracy and formalization [5, 12]. Large organizations succeed partly by being predictable; predictability is achieved through organizational inertia and the formalization of structures and processes [11]. Requirements must be elicited by specialized personnel, documented in formal and unambiguous terms, traceable back to their sources and forward to their implementations, and changed only with the oversight of a committee. Only through mechanisms such as these can large organizations deal with the challenge of communicating and controlling the requirements of their projects.

For small organizations, many of these activities are entirely unnecessary; there are plenty of documented cases of successful organizations that do fine without them [2]. This is largely because it is easier to share an understanding of requirements information with everyone involved on an as-needed basis. If a team is able to sort out its requirements problems by getting everyone together in the same room, it does not need to spend time creating documents that will soon become obsolete and might go unread [14].

2.2 Communication can be rich and robust

Organizations working on large projects must adopt some form of geographic distribution of effort. Even when the whole organization is located in the same area, communication between those of its members sitting beyond a short distance from each other is as low as if they were in different cities [1]. This is one of the factors that force large organizations to use inefficient communication mechanisms, such as requirements documents, to share project information.

In contrast, if the organization is small enough that it can work in a shared room or two, it is able to use much richer and pervasive communication dynamics [15]. This "radical co-location" has been found to lead to greater project efficiency and satisfaction [17], and it allows the organization to forego the creation and maintenance of unnecessary documentation. Note that the effects of

co-location benefit large organizations as well when they can partition projects in small sizes, but this is a natural advantage for small organizations.

2.3 Strong cohesion is possible

Small organizations can develop a strong group cohesion with relative ease. Group cohesion leads to increases in performance [3], partly because cohesive groups have lower coordination and communication overheads. They develop a shared vocabulary and a tacit understanding of each member's areas of expertise, enabling the maintenance of an efficient "oral tradition" within their teams.

This cohesion can even extend to members of customer organizations. Extreme Programming [4], for instance, advocates for the development of a close bond with customers, a bond that allows the team to understand the needs and culture of their clients intuitively and to resolve technical issues quickly.

2.4 Unscalable practices can be implemented

Many of the software development practices popularized in recent years, particularly those based on the Agile manifesto, prioritize co-located, cohesive teams over formalized processes. Arguably, part of the backlash against the Agile movement comes from the mismatch between its proposals and the formalized, geographically distributed, incohesive environments in which people attempt to apply them. It is possible to be Agile in large organizations, but it is not easy.

Agile techniques are a much better fit to smaller (less formal, co-located, cohesive) organizations. Story cards, backlogs, daily sprints, and other agile requirements practices depend on such an environment to prosper. They are tested, validated strategies to understand, prioritize, and track the requirements of a project, but they do not scale well—a problem for large, but not for small organizations.

3 Conclusion

We should not try to persuade small organizations to use the strategies we have devised for their larger counterparts. Some small organizations do apply them, perhaps out of a belief that it is the correct way to develop software, or a desire to emulate seemingly successful large firms. My position is that this is a mistake. Those strategies do not take advantage of their strengths; in fact they waste them entirely. Instead of proceeding down this path, we should welcome the opportunity to help these organizations identify their abilities, and to be explicit about the ways in which these abilities can be exploited to their advantage.

An important question remains: when does a growing organization cease to be small? That is, when do these strengths disappear? There seems to be a threshold after which organizational dynamics change, at around ten or twenty people [8]. Many organizations appear to have another qualitative jump at about one hundred and fifty [10]; there could be at least one more in-between these

two. To my knowledge, these thresholds have yet to be explored in the domain of software organizations. And it is possible that other determinants of size, such as the number of teams or the number and variety of customers, is more important than the number of employees for our purposes [13]. A better understanding of the construct of size and of the characteristics of software organizations of different sizes is an important step to advance our knowledge of the field.

4 Acknowledgements

I would like to thank Steve Easterbrook, Greg Wilson, Jon Pipitone, Neil Ernst, and Jonathan Lung for their insightful comments on this paper.

References

1. Thomas J. Allen. *Managing the Flow of Technology.* MIT Press, 1977.
2. Jorge Aranda, Steve M. Easterbrook, and Gregory V. Wilson. Requirements in the wild: How small companies do it. In *RE '07: Proceedings of the 15th IEEE International Requirements Engineering Conference*, pages 39–48, Delhi, India, 2007.
3. Daniel J. Beal, Robin R. Cohen, Michael J. Burke, and Christy L. McLendon. Cohesion and performance in groups: A meta-analytic clarification of construct relations. *Journal of Applied Psychology*, 88(6):989–1004, 2003.
4. Kent Beck. *Extreme Programming Explained: Embrace Change; 2nd Edition.* Addison-Wesley Professional, 2005.
5. Peter M. Blau and Richard A. Schoenherr. *The Structure of Organizations.* Basic Books, 1971.
6. Frederick P. Brooks. *The Mythical Man-Month.* Addison-Wesley, 1975.
7. U. S. Census Bureau. *Statistics retrieved from http://www.census.gov/.*
8. Theodore Caplow. Organizational size. *Administrative Science Quarterly*, 1(4):484–505, 1957.
9. Melvin E. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.
10. Robin Dunbar. *Grooming, Gossip, and the Evolution of Language.* Harvard University Press, 1996.
11. Michael T. Hannan and John Freeman. *Organizational Ecology.* Harvard, 1989.
12. Heather A. Haveman. Organizational size and change: Diversification in the savings and loan industry after deregulation. *Administrative Science Quarterly*, 38:20–50, 1993.
13. John R. Kimberly. Organizational size and the structuralist perspective: A review, critique, and proposal. *Administrative Science Quarterly*, 21(4):571–597, 1976.
14. Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003.
15. Gary M. Olson and Judity S. Olson. Distance matters. *Human-Computer Interaction*, 15(2):139–178, 2000.
16. E. Fritz Schumacher. *Small is Beautiful: A study of economics as if people mattered.* Blond and Briggs, 1973.
17. Stephanie D. Teasley, Lisa A. Covi, M. S. Krishnan, and Judith S. Olson. Rapid software development through team collocation. *IEEE Transactions on Software Engineering*, 28(7):671–683, 2002.