A SYNCHRONOUS TEAMWORK APPROACH TO SOFTWARE DEVELOPMENT

John Nash
Faculty of Administration

Timothy Lethbridge
School of Information Technology and Engineering

University of Ottawa
Ottawa, Ontario, K1N 6N5

jcnash@uottawa.ca

tcl@site.uottawa.ca

ABSTRACT

An active and synchronous teamwork approach to software development is suggested as a partial remedy to the shortage of programmers and quality problems faced by the software industry. The advantages and disadvantages of the approach are discussed. Examples of the use of the ideas are described.

MOTIVATIONS

The primary motivation for this note is the ongoing shortage of programming and related staff to create and maintain computer software and its documentation. The ideas we present here are, we believe, a partial remedy to the shortage, as well as a rational response to a number of management issues related to work of a creative nature. We are also motivated by the desire throughout the software industry to increase levels of quality.

Our ideas have been pursued in another direction in Nash and Nash (1997b) as part of the Visible Management initiative (Nash and Nash, 1997a).

THE SUGGESTION - TEAM PLAY

Our suggested approach to software development, indeed to creative work in general, is to have a large part of the activity performed by small teams. Moreover, the kind of teamwork we propose involves much closer co-operation than the word 'teamwork' usually implies in the software industry. Our sense of teamwork implies the members work together on each detail and discuss each point as the creative artifact is being produced.

As an example, the design and coding of a routine would be carried out by a two-person team, one of whom would be coding while the other makes notes that become the documentation.

Swapping roles from time to time, the team members cross-check and improve each other's work.

This approach has long-standing traditional support in many work-process situations. For example, airlines are flown by pilot and copilot in tight cooperation, and surgery is performed by a team working synchronously.

A persistent stereotype in the software industry is that of the solitary programmer, the "technonerd" who works alone. This stereotype has a considerable relationship to reality but is, we hypothesize, one of the obstacles to higher productivity in software development. Within the software engineering literature much emphasis is placed on team formation (see, for example, Lister and DeMarco, 1987). However, the use of teams are primarily seen as aiding the division of the work of a large project into manageable subprojects, and the streamlining of communication. The solitary programmer can happily work within such teams; each team member is still assigned a separate task and works independently most of the time (with periodic consultations and meetings). The synchronous teams we envisage are finer-grained and work together far more closely on a single task at a time.

We do not prescribe the exact manner in which synchronous teamwork be conducted. Several scenarios can be envisaged, from two seats at a single workstation to a cooperative session using some form of groupware. We discuss the synchronous teamwork approach here in terms of two-person teams, but do not preclude larger teams. The effectiveness of the approach for different types of work, different approaches to working together and different team sizes are issues we hope to be able to investigate in future.

An initial vision of synchronous teamwork in software development is as follows: The members first work together on requirements definition and design documents; one member at time develops figures and tables while the other develops the narrative text. When coding starts, one member may edit the comments while the other writes the code. In all cases, the team members will exchange roles perhaps every hour or so, or perhaps much more infrequently.

ADVANTAGES AND DISADVANTAGES

We hypothesize that the synchronous teamwork approach will have several important benefits:

> Firstly, and most significantly, there will be two workers rather than one familiar with each part of the work. This will have an impact on maintenance costs, since much time is consumed trying to understand the work of people who have left organizations. It will also likely lead to fewer errors caused by those who make changes without fully understanding a program.

Secondly, the approach will help prevent the introduction of defects at the outset, since each team member will be naturally checking the other members' work.

Thirdly, the time spent on quality assurance should be reduced. There is clear evidence (O'Neill, 1997) that inspections reduce the need for testing. With synchronous teamwork, the need for formal inspections themselves might also be reduced, since the other member of the team will be effectively performing inspections as the work progresses.

Fourth, the simultaneous creation of code and documentation - a difficult task for anyone working alone - is encouraged. In fact, the tendency to skimp or omit documentation altogether might be reduced. This may be particularly important for the design and maintenance documentation, since users rarely accept products lacking user documentation.

Fifth and finally, the approach may help to reduce time to market by shortening critical paths. Traditionally work elements such as code and user documentation have to be done sequentially by a single individual or by two separate individuals (Holland, 1997). With synchronized teamwork, a parallelism is introduced that results in both work elements being produced in a shorter total elapsed time.

On the negative side, we do occupy the time of two workers where, on initial observation, one would suffice. Our argument is that the both team members are, in fact, doing tasks that would have to be done anyway (i.e. inspection, documentation etc.) and that cost reduction due to fewer defects would more than outweigh any extra time consumed. However this hypothesis must be investigated.

Another potential problem is the necessity for highly effective interpersonal relationships at the working level between team members. Clearly many people do not "get along"; and new interpersonal problems may appear when people are required to work so synchronously together. Some workers may feel threatened by synchronous teamwork either because of feelings of inadequacy or due to difficulties in communicating and working in a team. These last two skills are often mentioned as desirable in job advertisements and are an issue of concern to the developers of training programs. Clearly managers cannot expect to assign people to teams without care, and only a certain percentage of software developers may have personalities that suit them for this kind of relationship. Moreover, re-assignments will almost certainly be necessary from time to time for reasons of human nature and personality.

An interesting issue about which we are unsure is whether the partners in synchronous teamwork must have the same level of ability. If experience or knowledge differ significantly, then synchronous teamwork might serve a training role, akin to a very close mentorship or apprenticeship arrangement. In such circumstances, however, some of the other advantages may be reduced or lost: The junior partner is unlikely to find defects, or write code documentation so effectively. One of us (TCL) has had experiences where the net output and quality level from such a relationship were, in fact, less than if the senior partner had worked alone. Despite this, smaller differences in experience level might, with the right partners, serve to train the junior partner while maintaining the other advantages of synchronous teamwork.

It is clear that synchronous teamwork will not work in all cases, perhaps not even most. However, we do feel that it is worth considering as a possible alternative to traditional approaches to producing software. We believe it will empower workers when they are suitably matched in ability and personality. And the workers themselves will know that it is working -- it cannot be imposed from outside.

EXAMPLES

The synchronous teamwork approach has been practiced for many years by one of us (JCN) in a number of creative activities of an academic nature. Most of these have involved the preparation of scientific or technical papers, and the usual co-worker has been Mary M. Nash. The teamwork approach came naturally and has resulted in a large body of joint work, though none is software.

A software example concerns a project, started in 1984, to develop a software system to allow scientists with a simple IBM PC or equivalent to carry out a wide variety of nonlinear least squares and nonlinear optimization calculations. This involved the development and testing of several hundred code and data modules and their documentation in two monographs (Nash and Walker-Smith, 1987; Nash and Walker- Smith, 1989). The conceptualization, development, testing, writing and editing were all carried out using synchronous teamwork. Reviews of the work were very positive, for example,

> As for the material on the disk, I found it to be both outstanding and extremely easy to use. The programming standards were obviously first-rate, and the documentation is excellent. (L. Zettel, 1989)

Another of us (TCL) has recently applied synchronized teamwork to the process of studying work practices (Singer, Lethbridge and Vinson 1998). Traditionally, the process of studying people performing tasks has been time-consuming and difficult because of the large amount of data gathered and the difficulty analyzing written notes and videotapes. Our synchronized shadowing technique involves two partners who work tightly together, each with their own lap-top computer, recording different types of details using a specially designed program: one person records the motivations, goals, successes, failures etc; while the other records the detailed actions. The computers are synchronized (identical clocks) so that the two types of data can be easily merged for later analysis. This method saves a tremendous amount of time compared to traditional approaches yet gathers data of similar quality. Its main innovation is tight synchronization of the work of two people, aided by computer.

DISCUSSION

Synchronous teamwork is not a panacea. It is, however, an approach that if applied to software development would likely offer advantages in increased productivity. These gains are achieved through defect prevention, better documentation and possibly the training or upgrading of staff.

It is clear that the nature of tasks that can benefit (or not) from synchronous teamwork, as well as the magnitude of any benefits and risks, need to be more precisely specified. The evidence in support of synchronous teamwork is as yet qualitative; measurements are needed. To this end we invite experience reports from those who have used similar approaches. We also solicit opinions about proposed details of the approach, its advantages and disadvantages, and expressions of interest for collaborative work to evaluate synchronous teamwork.

REFERENCES

DeMarco, Tom and Timothy Lister. 1987. Peopleware: productive projects and teams, New York: Dorset House Publishing Co.

Holland, Anton. 1997. Manual Labour: Documentation makes sense of technology, Government Computer, Canada Computer Paper Inc.: Ottawa, May 1997, pp. 18-22.

Nash, J.C. & Nash M.M. (1997a) The Visible Management System: management ideas with library principles, in Communication and Information in Context: Society, Technology and the Professions, (Bernd Frohmann, editor), Proceedings of the 25th Annual Conference, Canadian Association for Information Science, Toronto. pp. 124-130. Also University of Ottawa, Faculty of Administration Working Paper 97-38.

Nash, J.C. & Nash M.M. (1997b) Visible Management for Design, Programming and Other Creative Processes, submitted to the CSME Forum 98, Toronto, May 1998.

Nash, J.C. & Walker-Smith M. (1987) Nonlinear parameter estimation: an integrated system in BASIC, (with Mary Walker-Smith) Marcel Dekker Inc.: New York, 1987. Republished combined with the next item in electronic form by Nash Information Services Inc.: Ottawa, 1996.

Nash, J.C. & Walker-Smith M. (1989) Nonlinear parameter estimation: examples and software extensions, Nash Information Services Inc., Ottawa.

O'Neill, D. (1997) Issues in Software Inspection, IEEE Software, January 1997, pp 18-19

Singer, J., Lethbridge, T.C. & Vinson, N. (1998) Work Practices as an Alternative Method To Assist Tool Design in Software Engineering, submitted to CHI' 98

Zettel, L. (1989) Review of Nash and Walker-Smith, 1987, Computing Reviews 8901-0009, January 1989, p 53.