Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork

Salah Bendifallah and Walt Scacchi

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782

Abstract: The study and support of *teamwork* in upstream software development activities (e.g., specification, design) have been approached from a variety of perspectives. Those which address aspects of the division of labor typically focus on authority or communication structures. In this paper, we examine how teams of engineers develop software specifications, from a perspective emphasizing the division of labor in terms of the *work structures* themselves. We present a new typology of work structures and report on an empirical investigation of these work structures. We examine the teamwork process followed by each of five comparable teams of specification developers. The teams worked over a ten-day period with state-of-the-art specification resources to deliver functional specification documents meeting prescribed quality standards. Our data and analysis show the recurrence of various kinds of *shifts* in the teams' work structures. We discuss the resulting *patterns* of work structures and shifts and their implications. In particular, *separative* work structures were associated with improved specification teamwork efficiency, whereas *integrative* work structures were associated with improved specification product quality.

Keywords: Teamwork, Empirical Studies, Software Specification, Conway's Law.

1 Introduction

Twenty years ago, the participants at the Garmisch conference raised the issue of the relationship between group structure and system structure in large software development projects [32]. This relationship has been referred to as "Conway's law", after the researcher who had investigated it [13]. Conway pointed out the likelihood of a "mirror" relation between the structure of the design group and the structure of the resulting system design. This suggested the potential for effective management of large projects through judicious choice of group structure.

In large software development projects, upstream activities (e.g., specification or design) like downstream ones require teams acting in well-coordinated ways. They also require teams to act over extended periods of time within a particular organizational context. Many empirical studies of large projects done since the Garmish conference indicate that improving the efficiency and effectiveness of the earliest upstream activities is key to reducing development costs and increasing product quality [7,14,30]. Although Conway's conjecture still appears to provide a potential basis for such improvements, the problems of teamwork in upstream software development activities remain poorly understood [15]. It is not clear, for example, what division of work is appropriate for developing a large functional specification for an emerging software system, whose structure is typically unknown at the onset of the specification process. Concomitantly, the support provided (if at all) by current technologies for specification and design teamwork is generally limited (see for instance [9,10]).

Effective solutions require two complementary efforts:

1. Understanding, through empirical field studies and experiments, how small teams of software specifiers or designers develop specification or design products with state-of-the-art technologies in selected organizational and temporal contexts [4,15,16]; and
2. Exploiting this understanding to provide appropriate computer-support and teamwork guidance to assist in the cooperative development of high-quality specification or design products [11,12,20].

In this paper, we focus on an empirical study of specification development activities. We are concerned with how a specification team organizes its work over periods of several days to weeks in performing a prescribed project task constrained by quality standards [4]. To address this concern, we have analyzed specification teamwork in terms of *work structures*. Broadly defined, a work structure (WS) for a team task is the manner in which the components of the task are distributed among team members [25,33]. We view work structures as forming a third dimension of team structures, in addition to authority structures and communication structures, the two dimensions traditionally emphasized in studies of software engineering teams [17,29].

Recommended by: Pamela Zave

In looking at specification development by teams, we emphasize instead the teamwork tasks themselves and their division *in the course of project work* [4]. By examining the actual history of teamwork, we are able to highlight various types of organizational "breakdowns" [1] which contribute to the redirection of a team's work flow and to *shifts* in the team's work structures. Such breakdowns can lead to substantial losses of productive effort and unhappy tradeoffs between team effort and product quality.

The rest of this paper is organized as follows: After a brief review of related research in Section 2, Section 3 presents a taxonomy of work structures we have developed to understand specification teamwork. We use this taxonomy to examine the work process of five specification teams we have investigated through a field study in the System Factory project at the University of Southern California [35]. After describing our field study set-up in Section 4, we present our findings in Section 5. Our findings address the types of breakdowns the specification teams encountered, the consequent shifts in the teams' work structures, and the resulting patterns of work structures followed by the teams. Section 6 discusses these findings and their implications for efficiency and effectiveness in specification teamwork, and Section 7 closes the paper with a summary and suggestions for further research.

2 Related research

Our research is closely related to the few but varied empirical investigations of team performance and team structures in software engineering that have appeared since the Garmisch conference. Such investigations have been approached from a variety of perspectives [27,43], each of which provides particular insights for understanding team performance and teamwork processes and designing strategies and tools to support them.

The perspectives employed in investigating team structures differ in the aspects of the division of labor they address. Authority structures and communication structures have been the primary focus of most studies of programming teams, pioneered by Weinberg [42], Mills [31] and Baker [1]. Essentially, Mills' and Baker's "chief-programmer" (CP) team structure and Weinberg's "egoless-programming" (EP) team structure represent alternative distributions of authority and communication flows. These team structures were the subjects of further studies, such as by Scott and Simpson in 1975 [36] and by Mantei in 1981 [29]. Mantei compared CP and EP team structures along the authority dimension (controlled *vs* democratic) and the communication dimension (centralized *vs* decentralized). She further proposed a third, hybrid team structure, with controlled authority and decentralized communication, and discussed task situations suitable for each of the three kinds of team structures.

With their emphasis limited to authority and communication structures and downstream software activities, these studies could not address the underlying partitioning and allocation of development tasks among team members. In upstream activities, on the other hand, issues of task partitioning and allocation take on paramount importance. For example, the inherent composition of a software system's functional specification is usually unknown at the onset of the specification process; it may even remain unknown, due a desire to prevent partitioning biases, until work starts on developing an architectural specification.

Other studies have appeared which involved software engineering teams (see [3]) but did not consider the team structure among their independent variables. For example, Boehm and colleagues conducted an experiment at the University of California, Los Angeles, with seven teams of graduate software engineering students to evaluate the relative merit of the specifying approach and prototyping approach to software development [8]. Although their paper indicated that most teams adopted an EP-like structure, the main independent variable being investigated was the development approach itself, effectively precluding meaningful comparative results concerning team structures. Another such example is that of the "Cleanroom" experiment conducted by Selby and colleagues at the University of Maryland [37]. There, the 15 three-person teams adopted a CP team structure, but again, the focal independent variable was the development approach, whereby the first ten teams applied Cleanroom while the later teams applied a more traditional approach. Naturally, this also precluded comparative results concerning team structures.

More recently, Curtis and colleagues [15,16,24,28] have reported the results of field studies conducted by MCC researchers in the framework of the Leonardo project [30]. They are, like us, concerned with the actual processes of cooperation and breakdowns involved in upstream software development activities, but from a different perspective. The field study reported in [15], for example, spanned 19 software projects in nine companies to generate a model of how team consensus is reached through coalition formation within large design teams in the early stages of the specification process. This and their other studies thus address primarily the cognitive and communicative aspects of upstream software development activities. They focus on the analysts' and designers' mental models of the product in progress during the software activity, and on breakdowns which affect these models or arise from idiosyncracies and other incompatibilities in contending models.

In contrast, our analytical perspective is drawn from empirical studies of technical work organization in various settings [5,6,19,22,23,26,27,40]. In this perspective, we em-

[1] The word "breakdown" is used here as in [5,22,44], to denote an emergent disruption of the flow of work.

[2] This is the relationship traditionally addressed in the software engineering literature through the notion of " work breakdown structure" (see for instance [41]).

phasize the teams' work tasks themselves and how the organization of these tasks evolves in the course of the software development activity, in conjunction with progress on the product and changes in the project setting. This includes in particular paying attention to three sets of relationships: (1) among work tasks (i.e., task to task, task to subtask) [2], (2) between work tasks and people (i.e., who works on which task or subtask) and (3) among the workers as they relate to the work tasks (i.e., who works with whom on a task or subtask) [40]. In software specification teamwork, such task partitioning and allocation choices lead to several potential choices of work structures, which we present next.

3 Work structures

The first step in determining a WS for a team task is to consider whether the task is partitionable. A useful distinction in this regard is that between "divisible" and "unitary" tasks [38,39]:

divisible task A task that a team can readily divide into subtasks, each of which may be effectively carried out independently by a different subteam or team member. Each subtask of a divisible task can itself be further divided, unless it is considered unitary.

unitary task A task whose potentially distinguishable parts are sufficiently interdependent to warrant being treated as a whole for effective performance.

The other step in determining a WS for a team task is to consider whether the task is a product development task (or "product task") [20] or whether it is instead a "meta-task" [20]. Meta-tasks are tasks whose outcome is not a product or product-in-progress but a decision about managing the ordering and execution of product tasks (e.g., a plan, a schedule, a WS, etc.). The choice of work structures for a team's meta-tasks is constrained by the authority and communication structures adopted by the team.

- When the authority and communication structures are respectively democratic and decentralized [29] (as in an egoless-programming team structure [42]), meta-tasks are always considered unitary. Furthermore, in the spirit of a democratic authority structure, meta-tasks are readily handled through negotiation whenever their outcomes concern more than one team member.

- In contrast, when the authority and communication structures are (a) controlled and centralized [29] (as in a chief-programmer team structure [1]) or (b) controlled and decentralized (as in Mantei's hybrid team structure [29]), meta-tasks may be either unitary or divisible.

In the context of the System Factory project at the University of Southern California [35], project teams readily adopted an egoless-programming team structure for specification development. Consequently, in this study, our concern for meta-tasks is limited to those that are unitary and involve more than one team member; we thus leave out details about meta-tasks carried out independently by particular team members.

Starting from the distinctions between unitary and divisible tasks and between product tasks and meta-tasks, we have defined a taxonomy of work structures for software specification teamwork (see Figure 1). This taxonomy encompasses two classes of work structures based primarily on the distinction between unitary and divisible tasks, leading to six distinct types of work structures. We examine these in turn.

3.1 Work structures for unitary tasks

A unitary team task leads to four alternative types of work structures. The first two types correspond to situations where the unitary aspect of the task is handled by exploiting the potential synergy of collective interactions, such as in face-to-face meetings. The third type corresponds to situations where the unitary aspect of the task is viewed as warranting single-handed action. The fourth type corresponds to situations where the unitary task can be handled on several fronts in parallel.

- **Negotiative WS:** This is the case where the team task is a meta-task, and the result of the task is a *consensual outcome of negotiation.* The collective action involves primarily negotiation and the consensual outcome is a decision about the organization of work itself, e.g., a WS, plan, or schedule for a task. This is in contrast with the remaining five types of work structures, all for product tasks, where the outcome of team action is a prescribed specification product or subproduct.

- **Integrative WS:** This is the case where the final product of the task is a *consensual outcome of concerted action.* It is the outcome of collective action by a subteam or by the whole team, where the collective action involves continually concerted participation in the elaboration of a communal, consensual version of a prescribed specification product or subproduct. Accordingly, the evolving outcome of the overall team task is kept integrated at all times.

- **Delegative WS:** This is the case where the final product of the task is a *single-version outcome of individual action.* The task outcome results from the action of one individual team member to whom the task is entrusted.

- **Replicative WS:** This is the case where the final product of the task is a *multiple-version outcome of redundant action.* The task outcome includes many comparable and reconcilable outcomes of redundant parallel actions on a prescribed specification product or subproduct. This happens when a team elects to have two or more subteams (or individual members)

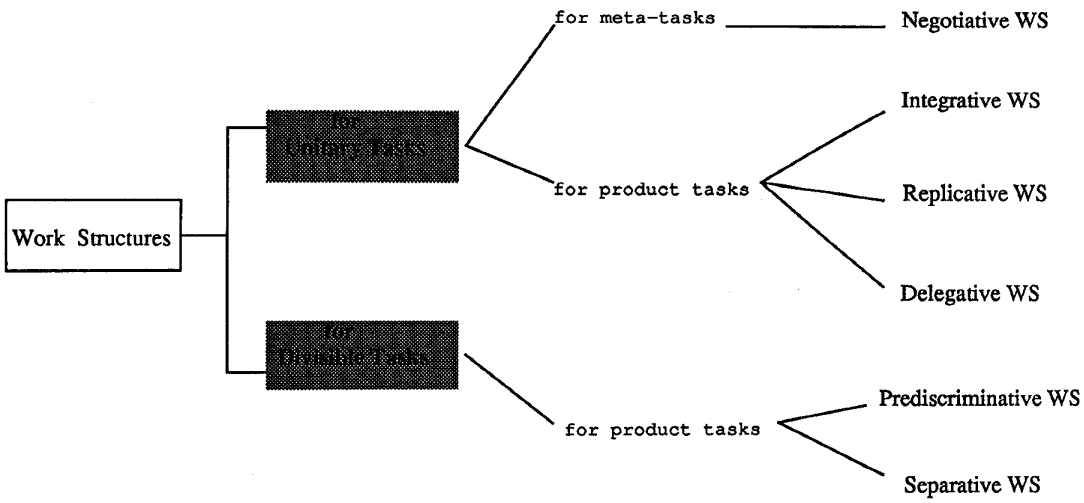

Figure 1: Taxonomy of work structures

simultaneously carry out the same task and thus obtain several candidate versions of the task outcome. Subsequently, these versions can be integrated into a comprehensive outcome by comparison, superposition or aggregation.

3.2 Work structures for divisible tasks

A divisible team task leads to two alternative types of work structures. In both cases, the objects of the task (e.g., functional specification document, formal functional specification) and the set of actions entailed by the task are divided. Each subtask involves a part of the whole object and its corresponding subset of particular actions, so that each subtask can be viewed as an idiosyncratic portion of the task. The final outcome of the whole task is obtained by rejoining the object partitions under their current *linking* constraints:

- **Prediscriminative WS:** This is the case where task partitioning is done along vested, *predetermined links.* Such links are those that reflect the document sectioning imposed through the functional specification document outline provided in the project's documentation standards. The cleavages among partitions are explicit in the required sectioning, and hence integration of the outcomes is typically not an issue. Integration merely involves inserting and juxtaposing the outcomes of the subtasks in their predetermined places, as indicated by the values of the links, i.e., the (sub)section numbers and/or headings [3].

- **Separative WS:** This is the case where task partitioning is done along *heuristically determined links.* Such links are not imposed by the project's documentation standards. They are instead created idiosyncratically by a team when the need is felt to superimpose a temporary partitioning on an object which, according to the project's documentation standards, should be considered monolithic and eventually delivered as such. Unlike in the case of a prediscriminative WS, novel conceptual cleavages must be perceived in the task (e.g., cleavages among functional requirements for, say, the task of developing separatively a formal functional specification) and carefully delineated. Subsequently, separation of foci is maintained *throughout* the duration of the task until they are resolved at integration time.

Each subtask of a divisible task in turn involves an appropriate choice among the five types of work structures for product tasks. Thus, some divisible tasks may be simple, whereas others may be complex:

- A simple divisible task is such that all its subtasks are carried out through delegative work structures. In this case, we do not indicate the subdivisions in our representation of work structures (as illustrated in Table 1). For example, a prediscriminative WS for a simple divisible task is merely indicated by the single label P in the table.

- Complex divisible tasks are such that at least one of the subtasks is carried out through a WS other than delegative. In this case, we explicitly account in our representation for the work structures for all the subtasks. This situation is illustrated in Table 1 in several instances of a prediscriminative WS. For example, the notation P(D, D, I) (see the sixth row of the table) indicates a prediscriminative WS with three task partitions, one of which is allocated to more than one team member and handled through an integrative WS.

[3] These links are typical of those used in software engineering hypertext systems [20], and in particular in the "assembly line" model of software development [21].

1. Review and understand the users' requirements document, in particular the desired functional requirements;
2. Understand the functional specification approach and functional specification standards prescribed in the project, and find and understand a specification library exemplar constructed using the approach and standards and relevant to the desired functional requirements;
3. Develop an informal functional specification which reflects the desired functional requirements;
4. Develop a consistent and complete formal functional specification; and
5. Develop a functional specification document satisfying the project's documentation standards.

Figure 2: **Prescribed work tasks for specification development**

Looking at this taxonomy immediately raises questions about the relative merit of adopting one type of WS over another for a given team task [4]. To answer such questions in the domain of software specification, we conducted a field study in which we investigated the work structures adopted by five teams of specification developers during a recent cycle of the System Factory project at the University of Southern California [35]. Before discussing our findings and their consequences, we will briefly present the set-up for our field study.

4 Field study set-up

We observed the functional specification development activities carried out by five teams using the Gist specification approach [2,11] over a period of ten calendar days. The teams had five to seven members and spent between 132 and 230 person-hours of teamwork to complete their tasks, depending on the team. We collected data through annotated participant-observation, [4] structured and open-ended interviews, a questionaire, and evaluation of the functional specification documents the teams produced at the end of the common timetable. At the onset of the specification development process, every team but one had access to a relevant exemplar (i.e., "reusable") specification that might be useful for completing their task.

Apart from this difference, all the teams had access to the same basic resources, had similar training in software engineering, and had to specify systems of comparable complexity (see [11] for examples of such systems). Also, all the teams initially adopted an egoless-programming team structure [42], whereby a team's authority structure is democratic and its communication structure is decentralized [29]. Furthermore, all the teams had access to the same specification tool [11] and guidance, with no prior exposure to the Gist specification development approach. The sequence of work tasks prescribed to the teams for developing a functional specification document, including a formal specification written in Gist, is shown in Figure 2.

[4]We had participant-observer positions in the project, in connection with our respective roles as project supervisor/teaching assistant and project manager/instructor. In these positions, we would generally observe, take notes, and infrequently answer questions, but otherwise leave the teams to act as they saw fit.

The same quality standards were prescribed to all the teams. The Gist language used by the teams [10,11,34] included the modularization construct "agent", which provided a suitable testbed for investigating work structures for specification teamwork. The agent construct allowed specification developers to circumscribe a particular system behavior and allocate it to a selected component of the overall system being specified [18,34]. The construct could moreover be exploited to devise potential ways of dividing functional specification development work among teammates wanting to follow a separative WS.

The first step in our investigation was to determine which work structures were adopted by the teams for their specification tasks. The second step was to explore the consequences of our findings. We summarize our analysis and findings in the next section and their implications in the subsequent section.

5 Patterns of work structures and shifts

The specification process followed by each of the five teams is shown in a succinct representation in Table 1. This coarse-grained description shows the chain of work tasks actually carried out by the teams (left column) and the work structures followed by each team for each of these work tasks. Space not permitting, we show in each case only the type of WS adopted, and leave out indications of mode of interaction, subteam composition, and other aspects of task performance. These details are described elsewhere [4]. We will refer to Table 1 in describing the specification teamwork process and discussing our findings.

5.1 Anticipated work structures and shifts

At the onset of the software specification process, the teams needed to evaluate available resources, both those prescribed in the project set-up and those available elsewhere (e.g., a team member's own personal computer and printer at home). The teams also needed to consolidate their authority and communication structure, as well as to establish a WS for carrying out each of the prescribed work tasks. This was the basis for managing their specification development work.

Team	T1	T2	T3	T4	T5
Size	6	7	7	7	5
Exemplar	no	yes	yes	yes	yes
pre-planning	N ⟶ R ⟶ I	N ⟶ R ⟶ I	N ⟶ R ⟶ I	N ⟶ R ⟶ I	N ⟶ R ⟶ I
planning	N ⟹ I I P (D, I, I) D R D	N ⟹ I I+ P (D, I, I) D R I	N ⟹ I I P (D, I, I) D R I	N ⟹ I ⟶ S S P (D, S⟶I, I) D D D	N ⟹ I ⟶ S S P (D, D, I) D D D
development of preliminary draft	I ➡ N ➡ R ➡ I	I+	I+	I+ ⟶ S+	I ⟶ S +
development of processable version of FFS	I+	I+	I+	S+ ➡ N ➡ D	S +
document write-up	N ➡ D	P+	P+	N ➡ P (D, S⟶I, D)	P (D, D, I)
document integration	D ➡ N ➡ I	D+	D+	D+	D+
document review	R	R	R	N ➡ R	D
preparation for delivery	D ➡ N ➡ I	I+	I+	N ➡ I+	D+

Legend:

* The labels D, I, N, P, R, and S stand for the kinds of WS adopted, repectively Delegative, Integrative, Negotiative, Prediscriminative, Replicative, and Separative;
* a WS label followed by + indicates the occurrence of shifts within the WS;
* a regular arrow (⟶) indicates an anticipated shift while carrying out a task (planned shifts between tasks are not explicitly indicated);
* a bold arrow (➡) indicates an unanticipated shift while carrying out a task.
* the double-lined arrow (⟹) points to the pattern of work structures and shifts originally planned by each team for the six kinds of development tasks;
* FFS stands for Formal Functional Specification.

Table 1: Patterns of work structures and shifts

For the initial pre-planning and planning tasks (see Table 1), we observed that all the teams adopted *the same types of work structures*. For pre-planning tasks, the teams followed a negotiative WS. Each team then shifted to a replicative WS so that the team members could evaluate, each on his or her own, their understanding of the prescribed functional specification approach. To consolidate the results of these initial accommodation tasks, each team then shifted to an integrative WS with a face-to-face meeting. Subsequently, for planning tasks, the teams shifted to a negotiative WS.

The outcomes of these preparatory self-management tasks for the teams included:

1. An elaboration of the originally prescribed work tasks (cf. Figure 2) into a sequence of major tasks, including those corresponding to distinct kinds of intermediate specification document subproducts (see the last six rows of the first column in Table 1); and

2. A sequence of work structures whereby each team intended to organize its cooperative execution of each major task.

As shown in the rows corresponding to *planning* tasks in Table 1, these sequences of work structures involved a number of anticipated shifts. In most cases, these shifts corresponded to the anticipated progression from one kind of task to another. In other cases (e.g., notably in the cases of teams T4 and T5), these shifts were anticipated within the same kind of task. Anticipated shifts resulted from each team's initial assessment of appropriate adjustments of its mode of cooperation to the intrinsic requirements of the different tasks needed to produce its deliverable documents. They also resulted in part from each team's initial assessment of circumstantial constraints in the project set-up, and in particular to the amount of scheduled interaction each team's members originally desired to maintain during the performance of each task.

5.2 Breakdowns and unanticipated shifts in work structures

Breakdowns [5,22,44] in the orderly execution and succession of anticipated tasks or subtasks often led the teams to make adjustments through *unanticipated shifts* in their work structures. We will look first at the various sources of breakdowns which emerged for the teams, and then at how the teams handled their work structures in recovering from these breakdowns.

Sources of breakdowns We found that breakdowns emerged from three kinds of sources:

1. *Idiosyncratic aspects of the project set-up:* for example, the short and somewhat pressured timetable of ten calendar days, the usability of some basic computing resources such as printers, the usability of the formal specification technique and tool provided, the reusability of specification library exemplars, and the distribution of team members' skills and interests;

2. *Changes in the project set-up in the course of the specification process:* for example, changes in the expected availability of some basic computing resources, the discretionary availability of some team members, the availability of accessory resources, and the distribution of team members' skills and interests; and

3. *WS adequacy:* for example, effects of work structures followed so far, or of originally anticipated work structures on team efficiency or quality of the products in progress.

Unanticipated shifts in work structures Upon the emergence of breakdowns affecting a team's work, the team members adopted a negotiative WS. They discussed desirable interventions with other project participants, with project management, or with support personnel. Then, if recovery required the team to accommodate to the new project situation, the team did so by making *unanticipated shifts* in its affected work structures.

The subsequent choice of a suitable WS depended on three sets of factors:

1. The intrinsic requirements of the task at hand;

2. The amount of scheduled interaction the team's members desired to maintain in the course of carrying out the task; and

3. The team's assessment of the time-quality tradeoffs warranted by the breakdowns.

5.3 Scope of shifts in work structures

Whether anticipated or unanticipated, all shifts in work structures involved adjustments either *in the type of WS* or *within a WS*. Anticipated shifts, such as those shown in the row corresponding to *planning* tasks in Table 1, involved primarily adjustments in the type of WS adopted to suit each of the major teamwork tasks. Depending on the team, unanticipated shifts included either mostly shifts in type of WS – in the cases of teams T1 and T4, or mostly shifts within a WS – in the cases of teams T2, T3 and T5.

Unanticipited shifts in type of WS Teams T1 and T4 experienced frequent unanticipated shifts in type of WS primarily because their choices of work structures for developing their preliminary draft turned out to be inadequate in the face of the time-quality tradeoffs warranted by some of the breakdowns the teams encountered:

- Team T1's integrative WS in the absence of an exemplar specification to help in the gradual development of its draft led to schedule slips. The team thus had to often shift away from the types of work structures it had anticipated to follow, in favor of work structures which would allow it to complete its tasks by the delivery deadline. In contrast, the potential inefficiency inherent in teams T2 and T3's choices of integrative work structures was mitigated by the availability of a specification exemplar.

- Team T4's adoption of a separative WS without sufficient forethought about the integration of its heuristic partitions led to quality problems. Fortunately, these problems were readily uncovered by one team member the team had elected to also act as quality monitor. Subsequently, the team had to often shift away from the types of work structures it had anticipated to follow, in favor of work structures which would allow it to improve the quality of its deliverable without trading off too much time. In contrast, team T5 did not monitor integration problems that could result from its choice of separative work structures. Consequently, it did not experience a need for shifts away from its anticipated work structures. Eventually

though, it lost in the time-quality tradeoffs by delivering a specification document of diminished quality.

Shifts within work structures Such shifts were made to accommodate breakdowns affecting various aspects of task performance. The main aspects thus affected included:

1. The manner in which the team realized the particular type of WS adopted for carrying out a task:
   (a) The composition (how many team members and who) of the subteams available for the tasks at hand; for example, some team members may not be available for a particular task because of prior commitments;
   (b) How team or subteam members could interact, e.g., through
      - discretionary interaction (in the case of any of the six types of work structures),
      - face-to-face meeting with a central display – VDT or chalkboard (in the case of integrative or negotiative work structures), or
      - face-to-face meeting with multiple VDT's (in the case of integrative work structures);
2. The inherent requirements of the task at hand:
   (a) The specification subproducts used as inputs to the tasks and their status (e.g., components of processable specification and the extent to which they could be integrated);
   (b) The resources specifically mobilized for the task, whether prescribed in the project (e.g., formal functional specification analysis tool, laser printer) or discretionary to the teams (e.g., personal printer) and
   (c) Pivotal subtasks of the major task at hand (e.g., reuse of formal functional specification fragment, maintenance of formal functional specification analysis tool).

5.4 Principal patterns of work structures and shifts

A careful examination of Table 1 reveals a variety of patterns in the sequences of work structures resulting from both anticipated and unanticipated shifts in each team's specification process. These patterns reflect the team's original choices of, and anticipated shifts among, work structures for the different kinds of tasks. They also reflect the number and kinds of shifts the chosen work structures actually underwent during the process.

For pre-planning and planning tasks (cf. Section 5.1), all of the teams followed the same pattern of work structures and anticipated shifts. However, for several of the development tasks, the teams anticipated and followed different types of work structures to carry out the same task. Such was the case for the tasks of developing the functional specification *itself* (i.e., developing a preliminary draft of the informal and formal functional specifications, and developing a processable version of the formal functional specification). For these specialized tasks of the process, both integrative and separative work structures (the two extreme types of work structures) were viewed as suitable for the tasks. Subsequently, the sequences of work structures and shifts followed for these tasks formed two basic patterns:

- a *separative* pattern in the case of teams T4 and T5; and
- an *integrative* pattern in the case of teams T1, T2 and T3.

Even for the document consolidation tasks, which are generic to the development of any document, roughly three patterns were followed from document write-up to preparation for delivery: one by team T1, another by teams T2, T3 and T4, and yet another by team T5.

6 Discussion

Our investigation has resulted in several insights about the specification teamwork process:

1. In many cases, the teams followed different work structures for different kinds of tasks. In most of these cases, the shifts in work structures that teams made when progressing from one kind of task to another were anticipated.
2. The teams also made shifts in work structures within a task. Some of these shifts were anticipated, but most were unanticipated. Unanticipated shifts were made in response to breakdowns in the project set-up affecting the task at hand.
3. Anticipated and unanticipated shifts occurred in either of two ways: a shift in the type of WS or a shift within a WS. The latter involved aspects of task performance such as the mode of interaction (e.g., face-to-face meeting *vs* discretionary interaction) and the resources mobilized for the task (e.g., provided resources *vs* accessory resources).
4. In many cases, the teams followed different work structures for the same kind of task. This resulted in a variety of patterns of work structures and shifts. Either of two predominant patterns was followed for the task of developing the functional specification itself: an integrative pattern or a separative one.

These findings have implications for computer support of the software specification teamwork process and for the intrinsic effectiveness of the process.

A major implication of our findings for computer support of specification teamwork is that this entails providing support for a variety of work structures and shifts. Potential support strategies should target all three aspects of this

variety: the different types of work structures, anticipated shifts, and unanticipated shifts. Our group has undertaken efforts in this direction [21,20].

The principal implication of our findings for the intrinsic effectiveness of the specification teamwork process concerns the relative merit of different patterns of work structures and shifts. We found that the two predominant patterns of work structures and shifts for functional specification development (i.e., integrative and separative) had consequences for the overall effectiveness of the process [4]. To obtain quantitative indicators of effectiveness, we collected data about teamwork efficiency and specification product quality. We looked in particular at the total number of person-hours expended by each team in the specification process (prorated to team size) and at the scores obtained by each team upon evaluation of its delivered document against the prescribed quality standards. The quality criteria we considered included for example the frequency of errors in the formal specification (determined by the automated specification processor [11]), and the structure of the delivered formal specification.

We thus found that integrative and separative patterns of work structures resulted in opposite tradeoffs between (1) efficiency of the process and (2) quality of the delivered specification documents. The breakdowns the teams encountered had magnified some inherent characteristics of each pattern of work structures. In spite of the short and somewhat pressured timetable, integrative teams readily invested more time than anticipated to reach consensus and obtain a high-quality product. In contrast, separative teams sought throughout to minimize face-to-face interaction [5]. Thus, even though such interaction could have helped resolve quality problems stemming from their initial partitioning and allocation of the functional specification development task, separative teams responded to the time pressure with a tradeoff detrimental to quality. This was eventually reflected in the structure of the formal specifications obtained by the teams, as separative teams delivered for example much less cohesive formal specifications than integrative teams [4].

Overall, our data and analysis suggest that *primarily integrative patterns will result in higher product quality, while primarily separative patterns will result in higher efficiency.* Thus, if a high quality specification is the desired outcome, then we should encourage and facilitate integrative work structures. On the other hand, if a specification must be developed rapidly, then we should encourage and facilitate separative work structures. These hypotheses are the subject of a follow-up study reported elsewhere [4].

[5] While primarily integrative teams approximated a unit ratio of time expended in face-to-face interaction to time expended individually, for primarily separative teams this ratio was halved.

7 Summary and conclusion

Twenty years ago, the software engineering community raised the issue of the relative effectiveness of different forms of team structuring in large projects. Subsequently, most empirical studies of the issue focused essentially on comparing alternative authority and communication structures suitable for downstream software activities. In upstream software activities, however, the most critical variable for team structuring concerns instead the underlying task partitioning and allocation among team members, i.e., the *work structure* itself.

In this paper, we have presented a taxonomy of work structures for software specification teamwork. We have then used this taxonomy in a comparative field study to examine the specification development process followed by five teams working under prescribed quality standards, including a standard for the structure of the formal specification the teams had to deliver as a section of their specification document. We presented the patterns of work structures the teams followed, the variety of breakdowns they encountered, and the shifts in work structures as well as the concomitant time-quality tradeoffs they made in response to these breakdowns. For the task of developing the functional specification itself, the teams followed primarily *integrative* or *separative* work structures. This choice had implications for subsequent time-quality tradeoffs, with opposite results for each of the two patterns. While integrative teams were the least efficient, they produced higher-quality results. On the other hand, separative teams were the least effective in terms of resulting product quality but were the most efficient. The difference in product quality was reflected, in particular, in the structure of the formal specifications the teams delivered. Whether this lends credence to generalizing Conway's law to work structures is naturally a question for further research.

Our findings also suggest that work structures represent a significant variable affecting software specification quality and productivity. Further, the influence of work structure variation may dominate that of the particular specification technology in use. Similarly, our findings suggest that specification technologies which either ignore the role of work structures or implicitly assume a particular work structure could have adverse consequences. They could potentially diminish specification quality or productivity if the work structures that specifiers adopt clash with that embedded in the technology. Such matters also require further study.

Acknowledgements

We thank George Bekey, Deborah Estrin, Pankaj Garg, Les Gasser, Elihu Gerson, Aziz Jazzar, Ron Rice, Leigh Star, Lucy Suchman and Terry Winograd for their comments on earlier reports of this work. We further thank Pankaj Garg, Les Gasser and Ron Rice for the extensive comments they have provided on a draft of this paper. Last but not least, we thank our anonymous reviewers for the insightful revisions they have suggested.

References

[1] F.T. Baker. Chief programmer team management of production programming. *IBM Systems journal*, 11(1):56–73, 1972.

[2] R. Balzer, N. Goldman, and D. Wile. Operational specification as the basis for rapid prototyping. *ACM SIGSOFT Software Eng. Notes*, 7(5):3–16, 1982.

[3] V.R. Basili, R.W. Selby, and D.H. Hutchens. Experimentation in software engineering. *IEEE Trans. on Software Engineering*, SE-12(7):733–743, July 1986.

[4] S. Bendifallah. *Understanding Software Specification Teamwork: An Empirical Analysis and Model.* PhD thesis, Computer Science Department, University of Southern California, Los Angeles, 1989. (Forthcoming).

[5] S. Bendifallah and W. Scacchi. Understanding software maintenance work. *IEEE Trans. Software Eng.*, SE-13(3):311–323, March 1987.

[6] F. Blanchard and A. Cambrosio. Disaligning macro, meso and micro due process: A case study of office automation in quebec colleges. In *Proc. Conference of Office Information Systems*, pages 118–125, Palo Alto, CA, USA, March 23-25 1988.

[7] B.W. Boehm. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[8] B.W. Boehm, T. Gray, and T. Seewaldt. Prototyping versus specifying: A multiproject experiment. *IEEE Trans. Software Eng.*, SE-10(3):290–303, May 1984.

[9] G. Bruns. Technology assessment: Paisley. Technical Report STP-296-86, MCC Software Technology Program, September 1986.

[10] G. Bruns, D. Bridgeland, and D. Webster. Technology assessment: Gist. Technical Report STP-369-86, MCC Software Technology Program, November 1986.

[11] A. Castillo, S. Corcoran, and W. Scacchi. A unix-based gist specification processor: The system factory experience. In *Proc. 2nd. Intern. Data Engineering Conf.*, pages 522–529, 1986.

[12] J. Conklin and M.L. Begeman. gibis: A hypertext tool for team design deliberation. In *Proc. Hypertext'87 Workshop*, pages 247–251. The U. of North Carolina, Chapel Hill, NC, November 13-15 1987.

[13] M.E. Conway. How do committees invent? *Datamation*, 14:28–31, April 1968.

[14] B. Curtis. By the way, did anyone study real programmers? In *Empirical Studies of Programmers (First Workshop)*, pages 256–262. Ablex Publishing Corporation, 1986.

[15] B. Curtis, H. Krasner, V. Shen, and N. Iscoe. On building software process models under the lamppost. In *Proc. 9th Intern. Conf. on Software Engineering*, pages 96–103, March 30 - April 2, Monterey, CA, USA, 1987.

[16] J.J. Elam, D.B. Walz, H. Krasner, and B. Curtis. A methodology for studying software design teams: An investigation of conflict behaviors in the requirements definition phase. In G.M. Olson, S. Sheppard, and E. Soloway, editors, *Empirical Studies of Programmers (Second Workshop)*, pages 83–99. Ablex Publishing Corporation, 1987.

[17] R. Fairley. *Software Engineering Concepts.* McGraw-Hill, New York, 1984.

[18] M.S. Feather. Language support for the specification and development of composite systems. *ACM Trans. on Programming Languages and Systems*, 9, April 1987.

[19] J. H. Fujimura. Constructing 'do-able' problems in cancer research: Articulating alignment. *Social Studies of Science*, 17(2):257–293, May 1987.

[20] P.K. Garg and W. Scacchi. On designing intelligent hypertext systems for information management in software engineering. In *Proc. Hypertext'87 Workshop*, pages 409–432. The U. of North Carolina, Chapel Hill, NC, November 13-15 1987.

[21] P.K. Garg and W. Scacchi. Composition of hypertext nodes. In *Proc. OnLine Information '88*, pages 63–73, volume 1. London, England, 6-8 December 1988.

[22] L. Gasser. The integration of computing and routine work. *ACM Trans. Office Infor. Systems*, 4(3):205–225, July 1986.

[23] E.M. Gerson and S.L. Star. Analyzing due process in the workplace. *ACM Trans. Office Infor. Systems*, 4(3):257–270, July 1986.

[24] R. Guindon, H. Krasner, and B. Curtis. Breakdowns and processes during the early activities of software design by professionals. In G.M. Olson, S. Sheppard, and E. Soloway, editors, *Empirical Studies of Programmers (Second Workshop)*, pages 65–82. Ablex Publishing Corporation, 1987.

[25] R.G. Hunt. On the work itself: Observations concerning relations between tasks and organizational processes. In E.J. Miller, editor, *Task and Organization*, pages 99–119. John Wiley & Sons, New York, 1976.

[26] A. Jazzar. *Understanding the Production and Consumption of Software Documents: An Empirical Analysis and Model.* PhD thesis, Computer Science Department, University of Southern California, Los Angeles, January 1988.

[27] R. Kling and W. Scacchi. The web of computing: Computer technology as social organization. *Advances in Computers*, 21:1–90, 1982. Academic Press, New York.

[28] H. Krasner, B. Curtis, and N. Iscoe. Communication breakdowns and boundary spanning activities on large programming projects. In G.M. Olson, S. Sheppard, and E. Soloway, editors, *Empirical Studies of Programmers (Second Workshop)*, pages 47–64. Ablex Publishing Corporation, 1987.

[29] M. Mantei. The effect of programming team structures on programming tasks. *Comm. ACM*, 24(3):106–113, March 1981.

[30] P. Marks. What is leonardo? Technical Report STP-141-86, MCC Software Technology Program, April 1986.

[31] H.D. Mills. Chief programmer teams: Principles and procedures. IBM Report FSC 71-5108, IBM Fed. Syst. Div., Gaithersburg, MD, 1971.

[32] P. Naur, B. Randell, and J.N. Buxton, editors. *Software Engineering: Concepts and Techniques.* Petrocelli/Charter, New York, NY, 1976. Proceedings, NATO Science Committee Conferences, Garmisch, W. Germany, October 7-11, 1968 and Rome, Italy, October 27-31, 1969.

[33] J.C. Naylor and T.L. Dickinson. Task structure, work structure, and team performance. *J. of Applied Psychology*, 53(3, Part 1):167–177, June 1969.

[34] W. Scacchi. Gist: An operational knowledge specification language. Research report, USC/Information Sciences Institute, Marina Del Rey, CA, April 1986. Draft.

[35] W. Scacchi. The system factory approach to software engineering education. In R. Fairley and P. Freeman, editors, *Issues in Software Engineering Education*, New-York, 1989. Springer-Verlag.

[36] R.F. Scott and D.B. Simmons. Predicting programming group productivity–a communications model. In *Proc. First National Conf. on Software Engineering*, pages 44–46, Washington, DC, USA, September 11-12 1975.

[37] R.W. Selby, V.R. Basili, and F.T. Baker. Cleanroom software development: An empirical evaluation. *IEEE Trans. Software Eng.*, SE-13(9):1027–1037, September 1987.

[38] M.E. Shaw. *Group Dynamics: The Psychology of Small Group Behavior (Third Edition).* McGraw-Hill, New York, NY, USA, 1981.

[39] I.D. Steiner. *Group Process and Productivity.* Academic Press, New York, NY, USA, 1972.

[40] A. Strauss. Work and the division of labor. *The Sociological Quarterly*, 26(1):1–19, 1985.

[41] R.C. Tausworthe. The work breakdown structure in software project management. In L.H. Putnam, editor, *Tutorial on Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers*, pages 210–215. IEEE Computer Society Press, 1980.

[42] G.M. Weinberg. *The Psychology of Computer Programming.* Van Nostrand Reinhold, New York, 1971.

[43] T. Winograd. A language/action perspective on the design of cooperative work. In *Proc. Conf. on Computer-Supported Cooperative Work*, pages 203–220, Austin, TX, USA, 1986. December 3-5.

[44] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design.* Ablex Publishing Corporation, Norwood, NJ, 1986.