The Importance of Team Skills for Software Development

by

Carolyn Tanya Wick
B.Sc., Simon Fraser University, 1995

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science

in the Faculty of Graduate Studies

Department of Computer Science

We accept this thesis as conforming to the required standard:

THE UNIVERSITY OF BRITISH COLUMBIA

Vancouver, Canada

December 1998

Copyright © 1998, Carolyn Tanya Wick
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purpose may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis in whole or in part for financial gain shall not be allowed without my written permission.

**THESIS TITLE:** THE IMPORTANCE OF TEAM SKILLS FOR SOFTWARE DEVELOPMENT

**DEPARTMENT OF COMPUTER SCIENCE**
**THE UNIVERSITY OF BRITISH COLUMBIA**
**2366 MAIN MALL**
**VANCOUVER, BC, CANADA**
**V6T 1Z4**
Abstract

Software does not just happen – it must be engineered systematically through the collaboration of individuals with necessary software development skills and appropriate tools. Similarly, effective software teams do not just happen – they too are built over time through the collaboration of individuals with appropriate team skills, tools and resources. This thesis outlines the need for superior software teams, reviews the literature on team performance and argues that a practical way to begin improving the effectiveness of software teams is in training software engineers in the skills necessary for effective teamwork. A list of fundamental skills for software development is presented, including skills for team leaders interested in building effective software teams, and general-purpose team skills benefiting all software team members. The most important team skills for software development are described in detail, including: project management skills, meeting skills, asynchronous communication skills, interaction skills, conflict management skills, group problem-solving and decision-making skills, skills for understanding the user or customer, team-building skills, leadership, and process awareness. The thesis concludes with a case study of an undergraduate software engineering team project course, in which several initiatives were taken to observe the performance of software teams and to learn more about teaching team skills to software engineers. Some special skills needed for effectively teaching team skills are identified, and recommendations are offered to educators for modifying the undergraduate software engineering curriculum, as well as to software professionals in industry for engineering their own effective software teams.
The Importance of Team Skills for Software Development – by Carolyn Wick
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneity</td>
<td>24</td>
</tr>
<tr>
<td>Team size</td>
<td>24</td>
</tr>
<tr>
<td>2.2.3 Challenge Factors</td>
<td>26</td>
</tr>
<tr>
<td>A clear and common purpose</td>
<td>26</td>
</tr>
<tr>
<td>Motivation</td>
<td>26</td>
</tr>
<tr>
<td>Attainable goals</td>
<td>27</td>
</tr>
<tr>
<td>Interdependency</td>
<td>28</td>
</tr>
<tr>
<td>2.2.4 Context Factors</td>
<td>28</td>
</tr>
<tr>
<td>The physical work setting</td>
<td>28</td>
</tr>
<tr>
<td>The performance evaluation and reward system</td>
<td>29</td>
</tr>
<tr>
<td>Other factors</td>
<td>30</td>
</tr>
<tr>
<td>2.2.5 Cohesiveness Factors</td>
<td>30</td>
</tr>
<tr>
<td>2.2.6 Conduct Factors</td>
<td>32</td>
</tr>
<tr>
<td>Underlying operational and teamwork processes</td>
<td>33</td>
</tr>
<tr>
<td>Best and worst software development practices</td>
<td>33</td>
</tr>
<tr>
<td>Process improvement</td>
<td>35</td>
</tr>
<tr>
<td>Process awareness</td>
<td>35</td>
</tr>
<tr>
<td>Flexibility</td>
<td>36</td>
</tr>
<tr>
<td>Organizational memory</td>
<td>37</td>
</tr>
<tr>
<td>2.3: SUMMARY OF SKILLS FOR SOFTWARE TEAM LEADERS</td>
<td>37</td>
</tr>
<tr>
<td>CHAPTER 3 SKILLS FOR SOFTWARE TEAM MEMBERS</td>
<td>39</td>
</tr>
<tr>
<td>3.1: CANDIDATE SKILLS</td>
<td>39</td>
</tr>
<tr>
<td>3.2: GENERAL-PURPOSE TEAM SKILLS FOR SOFTWARE DEVELOPMENT</td>
<td>42</td>
</tr>
<tr>
<td>3.2.1 Project management skills</td>
<td>42</td>
</tr>
<tr>
<td>3.2.2 Meeting skills</td>
<td>44</td>
</tr>
<tr>
<td>3.2.3 Asynchronous communication skills</td>
<td>47</td>
</tr>
<tr>
<td>3.2.4 Interaction skills</td>
<td>50</td>
</tr>
<tr>
<td>3.2.5 Conflict management skills</td>
<td>52</td>
</tr>
<tr>
<td>3.2.6 Group problem-solving and decision-making skills</td>
<td>55</td>
</tr>
<tr>
<td>3.2.7 Team-building skills</td>
<td>58</td>
</tr>
<tr>
<td>3.2.8 Skills for understanding the user or customer</td>
<td>59</td>
</tr>
<tr>
<td>3.2.9 Process awareness</td>
<td>59</td>
</tr>
<tr>
<td>3.2.10 Leadership skills</td>
<td>60</td>
</tr>
<tr>
<td>3.2.11 Other skills</td>
<td>60</td>
</tr>
<tr>
<td>CHAPTER 4 TEACHING TEAM SKILLS</td>
<td>62</td>
</tr>
<tr>
<td>4.1: HOW SHOULD TEAM SKILLS BE TAUGHT?</td>
<td>62</td>
</tr>
<tr>
<td>4.1.1 Integrate practice, theory, monitoring and assessment</td>
<td>63</td>
</tr>
<tr>
<td>4.1.2 Create a context to support effectiveness</td>
<td>65</td>
</tr>
<tr>
<td>4.1.3 Promote mastery</td>
<td>66</td>
</tr>
</tbody>
</table>
# List of Tables

Table 1. Software tasks that benefit from teamwork ......................................................... 3


Table 3. The Airlie Software Council’s “Project Caveats” of Worst Practices (1995) .............. 34

Table 4. Summary of the factors contributing to team effectiveness ................................. 38

Table 5. The Meeting Basics: Strategies for efficient and productive meetings .................... 45

Table 6. Guidelines for effective electronic asynchronous group communication .................. 49

Table 7. Functional roles of communication in group interaction ........................................ 51

Table 8. Procedures for teams .............................................................................................. 57

Table 9. Integrative problem solving process ....................................................................... 57

Table 10. Rules of brainstorming ......................................................................................... 57

Table 11. Initiatives for teaching software team skills in CPSC 319 (1996-97) ........................ 76
List of Figures

Figure 1. Software projects involve both operational and teamwork subtasks, processes and skills ....... 4

Figure 2. The "C"-rets to Success: Six key areas influencing team effectiveness ......................... 17

Figure 3. Stages of team development ........................................................................... 18

Figure 4. Competencies that members bring to a team ...................................................... 22

Figure 5. Important skills for software team members ....................................................... 41

Figure 6. Two-dimensional model for choosing appropriate conflict behaviour .................. 54

Figure 7. The experiential learning cycle ........................................................................ 64

Figure 8. Personality preferences recorded by students in CPSC 319 (February 25, 1997) ........ 89

Figure 9. Proportion of introverts to extraverts in CPSC 319 vs. in the population ............... 90

Figure 10. Problem-solving styles in CPSC 319 vs. in the Western Population ..................... 92

Figure 11. Future research in software team engineering will be inherently interdisciplinary .... 100
Acknowledgements

I would like to acknowledge both of my supervisors: Dr. Kellogg S. Booth, particularly for giving me so much freedom in pursuing my interest in software team processes, and Dr. Jeff Joyce, for ideas, inspiration and editorial assistance in focusing my topic for both academia and industry.

I am grateful to Peter Smith for letting me experiment with so many new ideas in CPSC 319, and to the many individuals interviewed for being so generous with their time and opinions, including: J. Brown (Harrison-Brown Technologies Inc., consulting for Motorola), L. Brown (Anderson Consulting), D. Canady (Microsoft), T. Dyck (Hewlett Packard), B. Freeman-Benson (Object Technologies International), J. Grattan (Southern Alberta Institute of Technology), J. Hyndman (Camosun College), S. Lapan (North Island College), J. Madhur (Rational Software, and 1998 INCOSE Symposium chair), M. Mantei (University of Toronto), K. Martiensen (Manager Human Factors, Canadian Airlines), J. Morrison (Microsoft), D. Powell-Williams (Chief Pilot B767, Canadian Airlines), S. Scroggins (Raytheon Systems Canada Ltd.), K. Toth (BC Software Productivity Centre), R. Tront (Simon Fraser University), C. Van Dyk (Greater Vancouver Regional District, Information Services), M. Weston (Camosun College), and D. Wick (Camosun College). I also wish to thank D. Wick, K. Wong, M. Greenwood, H. Mitchell, J. Nichols and A. Makepeace for reading rough drafts and helping me find the energy and focus to finish.

This research was supported in part by the Natural Science and Engineering Research Council of Canada (through a postgraduate scholarship), the BC Advanced Systems Institute, TeleLearning NCE, and the Media and Graphics Interdisciplinary Centre (MAGIC) at the University of British Columbia.

The Importance of Team Skills for Software Development – by Carolyn Wick
Chapter 1

Introduction

The accelerated growth of the software industry has created tremendous world-wide demand for highly skilled software technology workers. These workers must interact with one another to design, develop and maintain software. Poor communication and collaboration among the many people involved in software development can result in reduced product quality, products that are neither usable nor useful, duplicated and wasted effort, lower productivity, schedule overruns, increased stress, dissatisfied developers, higher employee turnover, and an inability to maintain long-term competitiveness. Effective software teams are less likely to have projects fail due to problems of communication and collaboration. As well, effective teams are increasingly important in alleviating the impact of serious universal software skill shortages, in handling the growing complexity of software, in improving software quality, and in helping software companies become and stay competitive.

This thesis argues that the effectiveness of a software team does not have to be left to chance. Team members and leaders with the right team skills, knowledge, tools and resources can improve the effectiveness of their software teams systematically over time. However, several surveys, publications and interviews reveal a shortage of team skills in the software industry, and a lack of resources for teaching them in undergraduate software engineering programs. To help mitigate these problems, this thesis describes a list of core team skills required for engineering effective software teams and proposes that the software industry as a whole will benefit from training software engineers in these basic elements of team effectiveness. Additionally, to demonstrate ways team skills can be taught to software engineers,
this thesis describes how several of the skills were introduced to students in an undergraduate software engineering project course.

The rest of this chapter defines some terms and concepts used throughout the thesis, outlines the role team skills play in software development, presents some arguments for why we need to be concerned with improving the effectiveness of software teams, and provides an overview of the other thesis chapters and of the research activities on which the thesis is based.

1.1 Definitions

The following terms are used throughout the thesis: team, software team, and software team skills.

1.1.1 Team

For the purposes of this thesis, a team can be defined as a distinguishable set of two or more individuals "working together toward a common goal, product, or solution that requires the sharing of expertise, knowledge, and ideas in a cooperative and interdependent fashion" [Nowaczyk, 1997]. This thesis is not particularly concerned with distinguishing between groups, work groups, teams and work teams, and hence the terms are used interchangeably throughout (however, if a distinction is required, Kinlaw [1991, Chapter 1] discusses ways that groups, teams and superior teams can differ, and Weinberg [1971] distinguishes between programming groups, programming projects, and program teams).

1.1.2 Software team

To limit the focus of this thesis to software development teams specifically, the following definition of a software team has been formulated: teams with a software systems design, process, or product focus.

1.1.3 Software team skills

Throughout the thesis, I identify and discuss various skills important for software engineers. I define
**software team skills** as those skills useful for software development that become more important when working with others than when working alone. Some of these skills are specific to software development, such as being able to track and control changes made to shared code, while others are transferable to many different types of teams. Some (described in Chapter 2) are more important for team leaders to understand, while others (described in Chapter 3) benefit all members of software teams. Teaching them also requires special skills (described in Chapter 4).

### 1.2 The role of team skills in software development

What makes team skills so important for software development? If software development were just coding, then for the most part one talented programmer working alone could develop software effectively. However, software development involves many more processes than just programming, as indicated in Table 1. It requires a variety of high-level skills, and benefits when multiple people with diverse abilities collaborate. Furthermore, many software systems are now “so complex that no one manager can comprehend the entirety” and “the challenge of complexity is not only large but also growing. The bang that computers deliver per buck is doubling every 18 months or so. One result is ‘an order of magnitude growth in system size every decade...’” [Gibbs, 1994c, p. 89] With software “exploding in size,” software teams and effective collaboration are no longer optional—they are mandatory.

**Table 1. Software tasks that benefit from teamwork [McConnell, 1996, pp. 275-6]**

- Developing and reviewing the project’s requirements.
- Developing the project’s architecture and design guidelines that will be used by the whole project.
- Defining aspects of the technical environment that will be used on the project (including the programming languages, compilers, source-code libraries, code generators, editors, and version-control tools).
- Developing coding standards that will be used by the whole project.
- Coordinating work on related pieces of a project (including defining interfaces between subsystems, modules and classes).
- Designing difficult parts of the system.
- Reviewing individual developers’ designs and code.
- Debugging difficult parts of the system [and resolving integration problems].
- Testing requirements, design, and code.
- Auditing a project’s progress.
- Maintaining software once it has been built (including responding to maintenance requests and making emergency fixes).
When a software team attempts any of these development tasks together, two types of underlying processes become important: the operational processes selected for completing the task at hand, and the generic teamwork processes governing how the team members interact. As Figure 1 shows, teamwork processes are often overlooked during software development and in the teaching of software skills, despite being integral to determining how effectively the team will complete its tasks.

As an example, consider the various development tasks in McConnell’s list. Many are highly interdependent. Different interpretations of the semantics of one module’s interface can radically alter the functionality of other modules. Changes in the software requirements can impact the chosen architecture or design, which if modified can severely impact the code and testing procedures. This interdependency can have huge consequences in terms of costs to fix defects: a requirements defect that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time [McConnell, 1996, p. 72]. Open, frequent communication is thus very important in addition to software-specific expertise for early detection and elimination of problems between interdependent tasks.

*Figure 1. Software projects involve both operational and teamwork subtasks, processes and skills*
Software teams are also prone to a number of unique pitfalls which strong team skills can help avoid. For example, projects can take teams from several weeks to years to complete. This is both a bonus and a curse to software teams. Unlike a team of pilots in an airplane cockpit forced to make life-critical decisions in seconds, software teams usually have sufficient time to carefully explore alternatives and think things through, and the long life cycle provides opportunities to return to decisions already made to correct mistakes and perfect solutions. However, this may encourage frequent changes, may promote indecision as teams put off making choices in favour of perpetually gathering more and more information and input, or may reduce the perceived criticality of each choice, resulting in poor decisions when members rely on the false security of being able to undo or redo decisions at a later date. Shaw points out that despite the relatively long development cycle for complex software systems, “when computers are embedded in light switches, you’ve got to get the software right the first time because you’re not going to have a chance to update it” [Gibbs, 1994c]. Strong problem-solving, decision-making skills, and an ability to eventually bring things to closure are thus essential in software teams.

Where the team of core software developers is large, skills for coordinating everyone’s work are obviously necessary. However, even where the team of core developers is small, team skills are still very important if not more so; the few developers will have to consult effectively with external customers, third-party software vendors, and other stakeholders whose backgrounds, cultures and demographic attributes may potentially be far more diverse. Extending the boundaries of membership to include project managers, marketers, technical writers, testers, and end-users may improve the quantity and quality of ideas and skills available, but may also introduce new communication problems. The members of this extended development team may not even speak the same language: there may be cross-cultural language gaps, as in English-Japanese collaborations, and there may be technology gaps, with communication difficulties translating between technical jargon and layperson lingo.

Finally, team skills are important because many software development problems are not technological. In a recent survey of 53 BC software companies [Toth, 1997], one question asked respondents to select their
top three software development problems from a list of 12 items. The results (ranked by percentage of respondents) are as follows (exact percentages not noted in the report, but supplied by Toth):

1. Schedule Overruns (45.2% of respondents felt this was one of their top 3 problems)
2. Shortage of Skilled Staff (41.9%)
3. Poorly Defined Requirements (32.3%)
4. Inaccurate Project Estimates (25.8%)
5. Team Communications (22.6%)
6. Migration to New Technology (22.6%)
7. Cost Overruns (19.4%)
8. Uncontrolled Changes (16.1%)
9. High Defect Rates (12.9%)
10. Poorly Defined Development Process (12.9%)
11. Inadequate Tools & Technology (9.7%)
12. Other (6.5%)

This data confirms what we see every day in the software industry. Most unsuccessful software development projects fail not because someone forgot the syntax for invoking a standard C++ routine, nor because they used an inefficient sort algorithm, nor even because they opted for structured analysis instead of an object-oriented approach. Most software engineering problems are in fact not “software engineering” problems at all, in the strictly technical sense. When projects fail it can often be traced to people issues, miscommunications, unclear expectations, or mismanagement. As Demarco and Lister [1987, p. 4] observed, “the major problems of our work are not so much technological as sociological in nature.” Effective teams can better handle the increasing complexity of software systems along with the inevitable social complexity associated with bringing together a group of talented people with diverse skills and experiences. Without members having effective team skills, software development groups are
more likely to be overwhelmed by the human interaction, communication and management issues that can
dwarf the already significant technical concerns.

These are just some of the reasons why team skills may be at least as important for software teams as for
other types of teams.

1.3 Is there a shortage of team skills in the software industry?

While it may already be obvious that team skills are quintessential for software teams, are they really
lacking among typical software team members? Is there a team skill shortage in the software industry?

Two things should be fairly obvious: most software development is occurring in teams already, and the
software industry is very successful. We could infer that most things must therefore be working well and
that software developers probably have sufficient team skills already. On the other hand, the software
industry's success may be in spite of relatively ineffective software teams, and the industry's rapid growth
and software's increasing complexity may actually mean that team skills are going to become more
important than ever in the near future.

In either case, continued progress depends on software engineers having certain prerequisite team skills
and experience to make their teams effective. How much practice and training do computer science and
engineering students receive in building effective teams? Learning objectives emphasizing essential team
skills like those listed in Chapter 3 are often absent from undergraduate computer science curricula. In
school we never receive a team report card. Despite being taught to share in kindergarten, from Grade
One onward through the completion of an undergraduate degree, if caught working together, we are
typically penalized for "cheating." Yet, when we finally get out into industry, we are expected to
suddenly collaborate effectively and demonstrate superior teamwork. Why are we not surprised by the
stereotypes that depict engineers and programmers as computer nerds who lack social skills, who prefer to
work alone, and whose "communication" skills are limited to a technical knowledge of TCP/IP and other
network communication protocols?

The Importance of Team Skills for Software Development – by Carolyn Wick
While software engineers deserve more credit for their skills than this cynical stereotype allows, we do see a definite lack of resources and training for teamwork skills in typical software engineering curricula. Common sense tells us that software engineers have not had many opportunities to develop effective team skills, and if the stereotypes are based on any element of truth, there is a good chance that the software industry does face a team skill shortage.

1.4 The importance of improving software teams

Why should we be concerned with finding ways of improving software team effectiveness by investing time and energy into teaching team skills to software engineers? Several reasons are identified here.

1. Team effectiveness is more important to software productivity than individual capabilities.

Assuming that demand for software technology continues to flourish, we can expect the salaries of top talent to skyrocket as companies try to attract hard-to-find skills. Unfortunately for British Columbia software companies, this may put some of the best developers beyond reach, as Canadian companies may be unable to compete successfully with higher salaries south of the border. Yet, even if the best people can be hired, research suggests that employing top talent may not in fact be the number one determinant of project success. A study of 31 software projects showed that group cohesiveness contributed more to productivity than did project members’ individual capabilities or experience, although individual capabilities were a close second [Lakhanpal, 1993]. A highly motivated cohesive team with merely average abilities can often outperform a group of experts if the experts cannot work together effectively. The reality of software development is that most projects are too complex to be completed by even the best programmers working alone. Software development requires a team effort. Because of this, and given the growing scarcity and expense of top software talent, success in the software industry will be determined over the next few years not just by the number of talented individuals a company employs, but also by the effectiveness of a company’s software teams. For some companies, increasing the
effectiveness of their existing teams may be a more realistic and economical goal than competing for the cream of the crop of new software developers.

2. **Effective teams can shorten development time.**

By increasing the effectiveness of software teams, organizations can raise labour productivity without necessarily hiring more people. How much more productive can effective teams potentially be? Research has shown dramatic differences in productivity across teams, with up to 5-to-1 differences in productivity among groups with different levels of experience, and 2.5-to-1 differences in productivity among groups with similar levels of experience [McConnell, 1996, pp. 12-14]. For a typical software project, this can translate into a difference of completing a project in either six months or one-and-a-half years.

3. **Effective teams can lead to better quality.**

A cohesive, communicative, flexible and efficient network of individuals working together has access to a wider range of information, ideas and abilities for completing tasks, is more likely to communicate problems early and openly, can respond more quickly to changing needs in the group, and can balance workloads more effectively so that all the team’s human resources are used to their fullest potential. This can result in fewer defects in the end product, fewer breakdowns, delays, or inefficiencies in the team’s process, and less variation in the outputs produced by each team member [Scholtes, 1988]. As well, specific practices that have been shown to improve quality, such as user-centred design, require skills more often found in effective teams: superior communication skills, an ability to manage the needs and expectations of diverse stakeholders, proficiency in establishing common goals, and a capacity to move from ideas to action in order to avoid the pitfalls of endlessly changing requirements and data collection.

4. **Effective teams satisfy members’ needs.**

Not only do software companies have to worry about being faster, cheaper and better simultaneously, but they also have to worry about keeping their software developers challenged and happy to keep employee turnover down and to improve their potential for attracting new talent. Fortunately, effective teams
represent a promising strategy for individual companies to attract and retain qualified people. Working in an effective team can be more stimulating and rewarding than working alone or on an ineffective team. Effective teams can be fun. Effective teams can give members a greater sense of control over factors influencing their work. Effective teams can provide members with more opportunities for growth, by matching individual interests and abilities more effectively with the team's tasks, and by adapting the team's goals and tasks over time to accommodate the changing needs and interests of members. Additionally, effective teams enable members to work on and complete larger projects than they otherwise could, breeding greater feelings of success and accomplishment which help build satisfaction. Higher satisfaction can in turn breed further success by increasing motivation and commitment, leading to lower turnover and even higher productivity and quality.

5. Effective teams have greater future capabilities.

As long as companies continue competing for talent amidst serious skill shortages, turnover will continue to be an unavoidable condition of software development no matter how satisfied the average worker becomes. To maintain long term effectiveness, software teams need to be able to carry on after individual members leave the group. Unfortunately, unlike individuals, teams are not born with the capacity to remember or learn over the long term, nor do they automatically have a form of persistent "team memory," other than the collective memories of team members. Creating an accessible and persistent form of organizational memory requires effort and discipline, but can set effective teams apart from ineffective teams. Effective teams with long-term memories have a greater capacity to learn from previous team experiences, are less likely to repeat mistakes, may be able to implement continuous process improvement, are more likely to develop stable repeatable processes that survive the loss of a team member, and can bring new members up to speed on the team's past activities and decisions more quickly [Humphrey, 1988; Paulk et al., 1993]. Additionally, by managing conflict and building cohesiveness, effective teams can ensure that members will have the desire and capability to continue working together in the future, saving considerable start-up costs of building new teams from scratch for each successive project.
These are some of the benefits enjoyed by effective software teams. The rest of this thesis examines more closely the skills that enable team members and leaders to realize these benefits for their respective software development groups.

1.5 Thesis overview

The rest of the thesis is organized as follows. Chapter 2 summarizes the literature on team performance, identifying skills useful for team leaders interested in building effective software teams. Chapter 3 extends this list to catalogue the skills that would benefit every software team member. Chapter 4 explores how team skills can be taught (as opposed to leaving them to be learned through trial-and-error), and suggests some skills for teaching team skills. Chapter 5 then provides a closer look at software teams in CPSC 319, the third-year software engineering project course at the University of British Columbia, where some skill shortages were observed and where initiatives were taken to explore issues of teaching team skills to software engineers. Lastly, Chapter 6 offers some conclusions, and the Appendices contain copies of materials used in the research along with extensive feedback from CPSC 319 students on their experiences working in software teams.

The material in this thesis builds on my background Bachelor of Science degree in Computing Science, on my extracurricular experience leading and facilitating small groups (particularly with Simon Fraser University’s Student Leaders, 1991-95), and on work experience in the software development industry at Microsoft, MPR Teltech, Hughes Aircraft of Canada, Asea Brown Boveri (Switzerland), Cerberus AG, and SFU’s Medical Computing Laboratory. Additionally, it is based on knowledge and experiences drawn from activities conducted in the M.Sc. program. These activities fall into four general categories:

1. Literature review – papers, books, videos and web-based documents were reviewed on the topics of teams, meetings, education and skill development, computer science curriculum, management skills, and software engineering best practices.
2. **Coursework, conferences, seminars, tutorials and workshops** – a variety of training activities developed background and expertise in areas related to the thesis:

- Software engineering best practices were surveyed through Dr. Kal Toth’s course “Software Engineering – Best Practices” (offered in Fall 1995 as UBC’s CPSC 537B), as well as through the Grace Hopper Celebration of Women in Computing Conference in San Jose, California (September 19-21, 1997).

- Two courses in organizational behaviour were taken at UBC: COMM 329 “Principles of Organizational Behaviour” (Fall 1996) and BAHR 507 “Principles of Negotiation” (Spring 1997).

- I attended workshops on participatory software design methodologies and other team-based software development practices, including Dr. Michael Muller’s “Participatory Design in the Software Development Process” (Vancouver, October 26, 1995), tutorials at CHI 96, Conference on Human Factors in Computing Systems (Vancouver, April 13-18, 1996), and a seminar by Gary Robinson on interactive team project management (Vancouver, October 22, 1998).

- Expertise in software process improvement was developed at SEPG 97, the Software Engineering Process Group Conference (San Jose, California, March 17-20, 1997), where I was influenced by numerous speakers, including McGrew, Bilotta, and Deeney’s tutorial on “How to Reach Level 2 With Small Teams,” Myers and Garcia’s “Change Agent Skillshop,” Wiegers’ “Creating a Software Engineering Culture,” Armento’s “Constructing an Effective Project Management Plan,” a panel discussion on “Overcoming Cultural Barriers to Software Process Improvement,” Mogilensky’s “Personality Types and Process Improvement,” and many others.

- The annual TeleLearning NCE Conference (Toronto, November 4-7, 1997) and UBC’s “Instructional Skills Workshop” (Summer 1996) introduced me to new teaching skills, methodologies and resources.
3. **Interviews and surveys** – the opinions and ideas of people from both industry and academia involved in software development, computer science education, and team-based industries beyond software engineering (such as the airline industry) were collected through questionnaires, formal interviews and informal conversations.

4. **Experimentation within a software engineering project course** – changes to UBC's third-year computer science course CPSC 319 were proposed as a way to introduce team skills training to the undergraduate computer science curriculum. As a teaching assistant and guest lecturer I was able to implement many of the proposed changes and observe students over the eight-month course.
Chapter 2

Team performance

A wealth of popular literature published recently for organizations, managers and team leaders promotes the use of teams. A glance at the shelves reveals every imaginable play on the word team, with titles such as *Empowered Teams*¹, *Team Building*², *The Team-Building Toolkit*³, *The Team Handbook*⁴, *The Wisdom of Teams*⁵, *The Tao of Teams*⁶, *Teams at Work*⁷, *How Teamwork Works*⁸, *Why Teams Don't Work*⁹, *Teaming Up*¹⁰, *Teams at the Top*¹¹, and *Team Talk*¹². These and many other books promote teamwork as a cure-all panacea for simultaneously developing faster, cheaper and better, hyped by the almost-too-good-to-be-true promise that the whole can be greater than the sum of its parts.

Unfortunately, team performance in many organizations has not lived up to the early inflated and often misunderstood claims; some companies who tried to implement teams without first understanding the factors that make teams work became disillusioned, as reflected in the popular Dilbert comic strip which makes fun of team building as yet another misguided and passing management fad [Adams, 1997, pp. 57-59]. Indeed, if teams are so great, why do so many software projects (almost always undertaken in teams) fail to finish on time, within budget or even at all?

This chapter defines what it means for a team to be effective, and summarizes the research literature related to team performance. Its main contribution is a systematic presentation of all the major factors

known to be correlated with team effectiveness that play a role in the performance of software teams. In so doing, it helps explain why many software engineering “best practices” do not guarantee success for all development groups; a software team’s methods and tools will affect performance, but they are only one factor among many variables that need to be understood to make software teams work. A better understanding of all the factors influencing team effectiveness described in this chapter would benefit anyone responsible for teams and anyone interesting in improving software team performance.

2.1 Qualifying team effectiveness

Team effectiveness ranges along a continuum where the degree of effectiveness can be calculated as a function of the group’s outputs which include:

- **The team’s performance** – the acceptability of the team’s product, process, or solution to those who receive or review it, where acceptability may be a function of output quality, quantity, time taken, errors made, and conformance to expectations;

- **The satisfaction of its members** – the degree to which members’ needs are more satisfied than frustrated by the group experience; and,

- **The team’s future potential** – the degree to which the members’ capability to work together in the future is maintained or strengthened.

All three of these areas are important and interrelated. The first of these, delivering usable and useful products on time and within budget to satisfy customers, should usually be a business’s number one concern. However, over the long term a team’s ability to do this consistently is intrinsically linked with the team’s ability to satisfy its own members’ needs and to maintain a potential for high future performance. For this reason, the rest of this chapter explores the factors that influence any or all of these areas of team effectiveness, rather than just team performance.
2.2 Factors that influence team effectiveness

What factors influence team effectiveness? The research literature repeatedly identifies many factors correlated with team effectiveness. I have categorized all the key influences into six areas shown in Figure 2: life cycle factors, composition factors, challenge factors, context factors, cohesiveness factors, and conduct factors. These areas (that I have termed the “C”-rets to success) were developed following a thorough review of dozens of books, articles, videos, and other materials summarizing what supposedly makes task-oriented teams effective.

Only when we understand all the factors influencing team effectiveness will we be able to identify and prioritize the skills software engineers need to control the effectiveness of their own teams.
1. **Life Cycle Factors:**
factors related to the team's maturity and current stage of development, as well as to how it formed, stormed, normed, performed or adjourned previously.

2. **Composition Factors:**
factors related to the team's composition (the team's size and characteristics members bring to the team including their competencies, status, demographics, and diversity).

3. **Challenge Factors:**
properties that relate to the task facing the team (e.g. its complexity) and characteristics of the corresponding goals set by the team.

4. **Context Factors:**
conditions of the external environment such as organizational reward structures, technology and resources available, level of stress, and so on.

5. **Cohesiveness Factors:**
the degree to which members become bound to the team by mutual attraction, coordinated or interdependent task behaviour, and shared beliefs.

6. **Conduct Factors:**
other factors affecting member behaviour, including the team's processes, norms, communication, leadership style, conflict management, meetings and role commitment.

**Team Effectiveness:**
- usable and useful quality products, satisfied customer
- satisfied members
- potential for high future performance

**Figure 2.** The “C”-rets to Success: Six key areas influencing team effectiveness
2.2.1 Life Cycle Factors

Since the mid-1960's, it has been believed that teams pass through a standard sequence of five stages [Tuckman, 1965]: forming, storming, norming, performing and adjourning. As Figure 3 illustrates, team effectiveness generally increases as teams progress through the stages, although the dashed lines show common points at which teams may fail in their development.

![Stages of Team Development](image)

**Figure 3. Stages of team development**

The types of task-oriented and social behaviours typically observed in teams differ from stage to stage, as do teams' potentials for productivity, conflict and failure, as follows.

**Forming**

During the forming stage, members must get acquainted and develop a positive working environment in order for the team to become effective. Members of effective teams tend to focus their efforts on defining goals, clarifying roles and expectations, developing commitment and acceptance, and developing procedures for performing their task [Laiken, 1994]. The stage is characterized by a lot of uncertainty about the group's purpose, structure, leadership and other member roles. Some behaviours and pitfalls common during the forming stage include the possibility that individual members might: (1) keep feelings to themselves until they know the situation, (2) act more secure than they actually feel, (3) experience confusion and uncertainty about what is expected of them, (4) be nice and polite, or at least not hostile,
and (5) try to size up the personal benefits relative to the personal costs of being involved with the group [Napier & Gershenfeld, 1985, pp. 459-460, as cited in Hellriegel at al., 1995, p. 274].

**Storming**

The storming stage deals with issues of power and control and is characterized by intragroup conflict and low work levels. Conflicts typically emerge over task behaviours, relative priorities of goals, who is to be responsible for what, and leadership. During this stage, members may feel as if they are “spinning their wheels” when task progress is impeded by inadequate skills in communication, decision-making, technical areas, and conflict management [Laiken, 1994]. Effective teams must learn to manage conflict during this stage (bring issues to the surface, legitimize concerns, invite input and feedback, expect and accept tension); they cannot effectively evolve into the third stage if they either suppress or withdraw from it. Sometimes an external facilitator can be brought in to help teams surface conflicts and manage them during this stage, and to help members develop team-building skills.

**Norming**

In the norming stage, social behaviours often centre on building cohesiveness, cooperation and group agreement. Mature task-oriented behaviours of members during this stage include openly sharing information, accepting different opinions, and supporting decision-making by seeking consensus while demonstrating willingness to compromise if necessary. One potential danger at this stage is that team members may swear allegiance to the team above the organization, as well as become unwilling to confront some issues for fear of “rocking the boat” and compromising cohesiveness [Wellins et al., 1991, p. 208]. The stage is called “norming” because it is when groups establish most of the acceptable standards or rules of behaviour (norms) by which they will operate. Some norms may be highly formalized, such as when a software team documents the coding styles to which its programmers will adhere. But most norms are informal, and include the assimilated team rituals, gestures, phrases and attire considered appropriate by members, and the acceptable patterns of punctuality, attendance, quality and
communication. For example, members of a given team may be expected to show unequivocal acceptance of ideas suggested by others at meetings, or by contrast, the norms could dictate that one always question and challenge any ideas that are put forward. Effective teams develop high-performance norms and increase cohesiveness during the norming stage.

**Performing**

The performing stage finds teams fully functional, with energies focussed on the task at hand instead of on getting to know and understand one another. However, productivity across teams at the performing stage can still vary dramatically depending on the level of cohesiveness, shared goals, unresolved conflicts, working atmosphere, and performance-related norms established in previous stages.

**Adjourning**

The final adjourning stage is experienced when groups wrap up their activities and high task performance is no longer the group’s top priority. Some teams have well-defined points of adjournment (like software teams tasked with shipping a new technology by a specific date), while other teams (such as quality assurance teams) may go on indefinitely and enter the adjournment phase more subtly when one or more key members leave the organization. Responses of team members during this stage vary between indifference, to satisfaction with the group’s accomplishments, to sadness over the loss of camaraderie and friendships gained during the team’s life [Robbins, 1993, p. 289]. Some members may even try to prolong the life of the team by avoiding the completion of certain tasks, by retaining ownership of some essential piece of information or object after the team has disbanded, or by preventing the project from finishing by continuously finding fault with the team’s product. Effective teams terminate their work positively and maintain a capability to work together again in the future.

Some authors have adapted Tuckman’s model of forming, storming, norming, performing and adjourning, such as Wellins et al. [1991, p. 188] who rename the first four stages: *Getting started*, *Going in circles*, *Getting on course*, and *Full speed ahead*. Such staged models provide a useful framework for describing
how teams mature and evolve over time and for structuring team-building initiatives and training in the skills needed to proceed from one stage to the next. However, while it is generally assumed that a group becomes more effective as it progresses through the first four stages, under some conditions, high levels of task-oriented conflict are conducive to high group performance and so there can exist situations where groups in the storming stage outperform those in the performing stage [Robbins, 1993, p. 289]. Similarly, not all groups proceed linearly from one stage to the next, some groups do not progress through all the phases, and others occasionally regress to previous stages.

2.2.2 Composition Factors

Undoubtedly, a team’s potential level of performance is influenced by who is on the team. Composition factors include the competencies that members individually bring to the group (such as their skills, knowledge and personalities), as well as the structural properties of the group collectively shaped by its members (including its status hierarchy, demographic heterogeneity, and size).

Competencies

Competencies are the underlying characteristics of an individual that are causally related to effective performance in specific situations. Figure 4 [Lawson, 1990, p. 25] presents a model identifying the competencies members bring to a team, including their learned behaviours, skills, knowledge, interests and motives, attitudes, personality characteristics, and physical traits.

The important thing to note about competencies is that some may more easily be changed than others. For example, members who do not possess the skills or knowledge needed to perform a task can often get up to speed with some additional training, whereas members whose interests, values, or personalities conflict with the behaviours required will more severely limit the team’s potential. For this reason, short-term team-building initiatives should not aim to change members’ values or personality preferences, but should instead aim just to develop the specific behaviours for teamwork and task completion. By contrast, teams should be formed with an eye to selecting members who already have compatible personalities and
other hard-to-change characteristics. Unfortunately, the inner characteristics are also the hardest to evaluate, although a number of new interviewing techniques and aptitude and personality tests are available for teams that wish to try [Falcone, 1997].

Finally, because the competency model implies that skills are acquired rather than given, it reinforces this thesis's premise that software professionals cannot be expected to have team skills unless they are taught them or are given opportunities to learn them through experience.

**Figure 4.** Competencies that members bring to a team
**Status**

Status is the socially defined position or rank given to team members by others. Status may be acquired formally through titles (e.g. "team leader") or amenities such as higher pay, higher grades in school, unique or preferred work schedules (e.g. exemption from working overtime or reserving weekends for family or other extracurricular commitments), windowed offices, or access to better labs and equipment. Status may also be acquired informally by such characteristics as age, education, gender, number of hours worked, access to information, networking power, experience, skills and other competencies, and anything else that members of the group evaluate as status-conferring. Inequitable status hierarchies have been shown to adversely influence motivation, productivity and the willingness to remain with an organization when individuals perceive a disparity between what they believe their earned status to be and what others perceive it to be [Robbins, 1993, p. 303]. For example, if a team's most capable and hard-working software developer feels she is being inadequately recognized and rewarded for her solutions, she will be motivated to restore equity by reducing her effort or by finding another company that better acknowledges her perceived status.

Effective teams maintain equitable status hierarchies. This can be difficult for software teams when some software components (graphical user interfaces) generate more easily identifiable and visually impressive outputs than other components (memory management routines, low-level error-handlers, or test drivers that are not even part of the final product). This can influence people’s perceptions of the significance of various accomplishments, and may lead to customers, managers and friends having different opinions from technical developers of the relative importance and difficulty of various tasks. One software manager acknowledged in an interview that this commonly creates status hierarchies that separate developers from testers, project managers, and users. Team members need skills for recognizing and amending inequitable status hierarchies.
**Heterogeneity**

The heterogeneity of a team is positively correlated with effectiveness. The more diverse the personalities, opinions, outside connections, skills and backgrounds of team members, the higher the probability that they will possess the needed characteristics to complete their tasks effectively, despite being simultaneously more conflict prone and less expedient due to having to introduce and assimilate more diverse positions [Robbins, 1993, p. 307]. However, research also suggests that this increased effectiveness applies only to groups that are “uniformly heterogeneous”, where everyone is moderately dissimilar from everyone else in the group. By contrast, large differences in demographic attributes such as age, gender, race, educational level, or time of entry into the group have been shown to predict turnover [Robbins, 1993, p. 308]. For example, if an all-male software team with similar social, cultural and educational backgrounds wanted to increase its heterogeneity by hiring a female, we could predict that this discontinuous introduced heterogeneity would more likely result in the lone female leaving than in the team’s effectiveness improving.

**Team size**

Team size has been shown to impact performance in different ways depending upon the type of task in which the team is engaged [Fleishman & Zaccaro, 1992, p.37]. Generally speaking, larger groups are better for “disjunctive” tasks like fact finding, brainstorming and testing for which the group’s effectiveness is not impeded by the performance of weak members, but instead, is limited only by the most capable members of the group and the team’s collective expertise. Conversely, smaller groups are better for “conjunctive” tasks like action taking, implementation and decision making which require judgment, consensus, and contributions from all members. Since software development involves both types of tasks, the optimal size for a particular team will vary according to the project and the capabilities of members. As well, effective teams may choose to break into subgroups for some tasks or recruit additional people to join the group temporarily for other tasks. Some things to consider when defining the boundaries of membership are as follows.
Larger teams have more available energy to put into the task, making it psychologically less daunting to move the possible completion date closer. However, teams that contain more members than needed to supply the necessary task and interaction skill may suffer the opposite effect on motivation if the task is perceived to no longer be challenging enough. Furthermore, research shows that individual group members often decrease their efforts as group size increases. This "social loafing" effect is the tendency of group members to do less than they are capable of individually and has been demonstrated on both physical and cognitive tasks [Robbins, 1993, p.306; Fleishman & Zaccaro, 1992, pp.37-38]. Fortunately, this effect can be eliminated even in large teams, but only when member outputs are made identifiable, when the task is made sufficiently difficult, when members have high personal involvement, and when the task orientation of the group is enhanced [Fleishman & Zaccaro, 1992, p. 38].

"The more, the merrier" is true in that a larger group may socially have more fun working together and may generate additional energy to put into the task through healthy internal competition, support and mutual motivation. However, members are also likely to obtain fewer personal satisfactions from the team experience with more people involved (especially when individual accomplishments are more likely to go unnoticed, when group cohesiveness decreases, and when interactions are likely burdened by more conflicts) [Seaman, 1981, pp. 42-45]. As well, Hellriegel et al. [1995, p. 281] caution that "twelve members probably is the largest size that enables each member to interact easily with every other member."

Robbins [1993, p. 306] recommends a compromise: "Having an odd number of members eliminates the possibility of ties when votes are taken. And groups made up of five or seven members are large enough to form a majority and allow for diverse input, yet small enough to avoid the negative outcomes often associated with large groups, such as domination by a few members, development of subgroups, inhibited participation by some members, and excessive time taken to reach a decision."

Finally, the benefits of having a large or small team need to be balanced against the disadvantages of increasing or decreasing the size of an already existing team. Brooks [1995] provides a critical look at the
mythical man-month. He shows that even if members maintain a constant output of effort as a group grows, the increased communication overhead and time needed to bring new members up to speed mean that the added human resources will not translate linearly into productivity increases.

2.2.3 Challenge Factors

In addition to the preceding composition factors, a team’s productivity is influenced by what the team has to accomplish. Do members have a clear and common sense of purpose? Is the team’s task sufficiently motivating? Do members believe their goals are attainable? Is the level of interdependency among member roles appropriate to the team’s interaction skill? These characteristics of the team’s challenge can each influence effectiveness.

A clear and common purpose

Group members must know where they are going [Harvey & Drolet, 1994]. All require a shared understanding of the team’s purpose and desired outcomes. Because members usually enter a group with some preconceived notion of what is to be accomplished and how this task might be achieved, aligning team members’ diverse perceptions of individual and team goals can potentially be laborious and time-consuming. However, establishing a clear and common purpose provides direction for a team, eliminates confusion, provides criteria for evaluating group effectiveness, and makes it easier to recognize when a task is complete.

Motivation

To complete their tasks, members must want to get there [Harvey & Drolet, 1994]. One way to motivate them is to ensure that their tasks are sufficiently challenging given the number of people on the team, their collective competencies, and their allocated time and resources; if goals are too easily accomplished, members may lose interest in the team. Another way is to involve all members in the initial goal setting process so that members get a chance to “buy into” the team’s purpose and influence the team’s goals.
with their personal objectives and expectations. Members will subsequently feel that they are working
toward their own goals, which should be more motivating than working toward goals that have been set
arbitrarily by someone else [Seaman, 1981, p. 105]. Still other motivational aspects of group tasks are
suggested by Hackman [1987, p.324], who indicates that members are likely to be motivated and satisfied
with their work when: (1) the group task requires a variety of relatively high-level skills; (2) the group
task is a whole and meaningful piece of work, with a visible, identifiable outcome; (3) the group’s
outcomes have significant consequences for other people and groups; (4) the task provides group
members with substantial autonomy in deciding how to do the work; and (5) work on the task generates
regular, reliable performance feedback. These are just some of the characteristics of the team’s challenge
that can motivate members to work especially hard.

Because software development requires high-level skills and usually results in a visible, identifiable
product, focusing on the task itself can have considerable motivating potential. However, teams
developing an exciting new technology may be more motivated than teams whose primary purpose is
maintenance; similarly, goals focussed on product features may motivate more than goals centred around
meeting a particular delivery schedule. Furthermore, unlike a team of construction workers building a
house, software developers do not always have the benefit of regular performance feedback from the task
itself. Progress visibility does not come automatically, especially for teams who, instead of following an
iterative or evolutionary approach, use development methodologies that delay tangible results until the
end when everything is integrated. Thus, it is particularly important that software teams find ways to
obtain feedback and measure progress so they can stay motivated through the long development lifecycle.

Attainable goals

In addition to wanting to get there, members must believe they will get there [Harvey & Drolet, 1994].
Effective team goals are appropriate, realistic, challenging, and yet not overly ambitious; otherwise, they
create a situation that inevitably leads to disappointment, frustration, and failure [Seaman, 1981, p. 105].
Furthermore, lack of faith among members in their team’s capability to complete the task at hand can lead
to reduced motivation and apathy [Seaman, 1981, p. 60]. Attainable goals are therefore important for determining whether members' needs are more frustrated or satisfied by the team experience.

**Interdependency**

Finally, team members must have a reason to work together [Harvey & Drolet, 1994]. Effective teams choose paths that include sufficiently complex tasks that must be done in cooperation and not just side by side. Interdependent tasks give members a reason to be a team, help hold the team together, build trust, cohesion and collaboration, and enable the team to more fully benefit from the diversity of its members. However, for a team to be effective, the level of interdependency of the team's tasks must be appropriate to the members' interaction skills. High interdependency can propel teams to higher levels of effectiveness since members must develop superior communication, leadership and conflict management skills in order to succeed; nevertheless, groups with poor communication, weak leadership, and high levels of conflict can still be considered effective when assigned simple tasks with little interdependency [Robbins, 1993, p. 310].

### 2.2.4 Context Factors

Many important factors associated in the literature with effective teams are properties of a team's context, the external environment and conditions imposed on the group. Often these conditions are difficult or impossible for team members to influence or control and yet they frequently determine aspects of a group's behaviour. Two examples of context factors include the team's physical work setting and its performance evaluation and reward system.

**The physical work setting**

Should team members work in cubicles or offices? Are windows really necessary? Which is more disruptive: a telephone call or e-mail? Should teams invest in ergonomically designed keyboards and chairs? DeMarco and Lister [1987] discuss how some of these aspects of the physical work setting can
influence software team productivity. As it turns out, two significant factors influencing software team performance are the proximity and privacy of members. Of particular note is the following paradox: the closer co-workers are located to each another, the more frequently and easily they will be able to communicate and collaborate; however, unless a low-enough density of people is maintained in the workplace, productivity losses due to noise, stress and other distractions may outweigh the benefits of having team-mates situated proximal to one another. Koch [1997]'s research into the coordination of hardware and software development teams supports this in part by showing that the distance between teams was crucial to project success (the closer the hardware and software teams were to each other, the more likely they were to meet project cost goals). Based on these observations, teams and leaders should do whatever they can to improve the proximity of all stakeholders involved in a project, as long as it does not come at excessive cost to the team members' privacy or autonomy.

The performance evaluation and reward system

The performance evaluation and reward system can also both facilitate and impair performance. In several laboratory studies, subjects who believed that an experimenter could evaluate their performance, subjects who were able to compare their performance to a standard, and even individual group members provided with a standard against which they could compare their group's performance (but not their individual performance) all increased the quantity and quality of individual performance over people who believed there was no potential for evaluation [Penner & Craiger, 1992, p. 67]. That having been said, perceived inequities between potential rewards and the time, skill, effort and other costs involved in performance may lower members' motivation to perform. As well, some research studies indicate that potential evaluation of an already highly motivated individual by an observer, particularly a potentially negative evaluation, facilitates performance on simple tasks, but impairs performance on difficult tasks, possibly because cognitive overload may result from focusing too much attention on others' expectations in addition to concentrating on the task itself [Penner & Craiger, 1992, p. 62]. Evaluation and reward systems can further impair performance if they are intolerant of errors and mistakes (discouraging risk-
taking and creativity), or if they promote a "scarcity mentality" within the team where rewards are perceived to come out of a limited pie (encouraging competitive behaviours, domination of the task activity, and the withholding of information from other team members to maximize individual rewards at the expense of the group's goals).

How can teams use the evaluation and reward system to facilitate performance? Productivity will be highest when members are convinced that their rewards will be greatest if they work hard to accomplish the group task. Teams should assess the external potential payoffs for completing the task, and if need be, institute internal payoffs themselves (such as praise, increased status within the team, and tangible internal rewards) to balance it out [Seaman, 1981, p. 46]. In particular, teams will do well to ensure that the payoffs reward teamwork, in addition to productivity, for the group to develop into an effective team; otherwise, members will perceive meetings, team building, collaboration, social activities, and process improvement initiatives to hold little or no value.

Other factors

Context factors influencing team effectiveness are not limited to the physical work setting and evaluation system. The level of stress members experience in completing their tasks, their access to technology, and the team's autonomy (does it have the authority and power to implement its own decisions as well as make its own decisions about what it will implement?) all can influence team effectiveness. Furthermore, the larger organization's culture, personnel selection process, formal regulations, strategy and resources all play a role [Hellriegel et al., 1995, p. 276; Robbins, 1993, pp. 292-293]. It would seem that effective teams do need to credit a supportive context for part of their success, which sponsorship could facilitate.

2.2.5 Cohesiveness Factors

Often, when people reminisce about great teams of which they were once a part, they recall how much they enjoyed being on the team and how well the team jelled. Cohesiveness is the degree to which group members are bound to the group by mutual attraction, interdependent task behaviour, and shared beliefs.
Contrary to popular opinion, cohesiveness does not in and of itself guarantee effectiveness. However, it is both a major contributing factor and an outcome of high productivity in a relationship moderated by the team’s performance-related norms.

The relationship between cohesiveness, performance norms and productivity works as follows. The more cohesive the group, the more its members will follow its goals. Where performance-related norms are high (such as when team members value high output, quality work, and cooperation with individuals outside the team), a cohesive group will be more productive than a less cohesive group. However, where performance norms are low, increasing cohesiveness has been shown to actually lower productivity [Robbins, 1993, pp. 312-313].

Assuming then that a team has established high performance norms, it can become maximally effective by becoming highly cohesive. What factors affecting cohesiveness can team members influence? Just as cohesion influences productivity, so can productivity affect cohesion; effective teams can increase cohesion by building up a history of previous successes (even subtle accomplishments like clarifying the common direction or reaching consensus on an issue can be significant), and by recognizing and celebrating individual and team accomplishments through publications, presentations, praise and victory parties [Williams, 1993]. The amount of time members spend together also positively affects cohesiveness, and so teams can attempt to become more cohesive by meeting and seeing each other frequently, by increasing the physical proximity of members, and by increasing opportunities for interaction (either face-to-face or through electronic means). Similarly, members are more likely to be committed to the team when the team has a distinct identity, when the team’s purpose is clear and valued, and when members have fun together [Williams, 1993]. Relationships between cohesiveness and group size are described by Robbins [1993, p. 311]. For instance, cohesiveness decreases as group size increases, since interaction with all members becomes more difficult as does the ability to maintain a common goal; keeping groups small will therefore help maintain cohesiveness. Fortunately, the more difficult it is to get into a group, the more cohesive the group becomes; some groups may thus be able to...
increase cohesiveness by establishing criteria for joining the team and by involving all members in the selection and initiation process.

One final consideration to note when attempting to improve effectiveness by increasing cohesion is that the best approach depends on the team’s purpose. For social groups, out of the many ways of increasing cohesiveness, heightening interpersonal attraction will have the greatest impact on a group’s productivity; for ideological or interest groups such as religious or political organizations, shared beliefs are most relevant; and for task groups, task coordination and interdependency are most important [Driskell & Salas, 1992, pp. 118-119]. Thus in software teams, for example, structural task integration will be more important than how well members like one another.

2.2.6 Conduct Factors

While teams can succeed in spite of poor composition, an unclear purpose, or unsupportive context, they obviously cannot succeed at all if they do nothing. Moreover, a highly cohesive team with an ideal composition, challenge, and context can still fail if it does the wrong thing. Obviously then, effective team conduct is important for success, and the underlying processes guiding that conduct will influence the team’s effectiveness. Groups may have the potential to surpass individual performance but will not successfully achieve that potential if group processes are counterproductive. As Steiner [1972, p. 9] observed:

\[
\text{Actual Productivity} = \text{Potential Productivity} - \text{Losses Due to Faulty Process.}
\]

In theory, teams can be made more effective by improving or eliminating the “faulty processes” that are causing losses in productivity. As tasks become more complex and interdependent, the conduct of members becomes increasingly important in determining whether or not a team will successfully achieve its goals and how much potential productivity will be lost due to faulty processes.
**Underlying operational and teamwork processes**

The candidate processes for improvement fall into two camps: the operational processes selected for completing the task at hand, and the generic teamwork processes governing team dynamics and behavioural interactions among team members. In most teams, a few of these processes may be well defined and optimized, but the majority of processes governing team conduct are usually implicit, ill-defined, and dependent on the people performing them. This can make it difficult to identify ways of improving performance, and increases the risk of failure for teams tackling challenging problems over long periods of time where there is a possibility of losing key players part-way through the project.

**Best and worst software development practices**

Given that processes influence whether a project succeeds or fails, teams may benefit from knowing which processes other teams adopted on successful or failed projects. In the software field, numerous researchers and practitioners have compiled lists of best and worst practices for software development. Two examples of such lists are provided in Tables 2 and 3, the first being a comprehensive summary of some of the best “rapid development” practices which are each described in more detail in McConnell’s book [McConnell, 1996], and the second taken from the Airlie Software Council’s recommendations of practices to avoid based on reviews of numerous failed software projects [Yourdon, 1995].

Unfortunately, it is very difficult to correlate any specific process or behaviour with success for all types of software teams or projects, largely because the tasks and conditions facing teams are unique and diverse, and many successful practices are mutually exclusive. Yourdon [1995, p. 12] cautions that no particular practice or combination of practices can guarantee success or failure; best practices “are not intended to be universal truths to be imposed unthinkingly upon your development organization; you should use them as a starting point, match them against your own organizational experience, and devise your own customized list.” Nevertheless, a team’s processes and behaviours – the conduct factors – form the single most important area for team members themselves to focus on improving because they are the factors most within the team’s locus of control.

<table>
<thead>
<tr>
<th>Fundamental Software Engineering Practices</th>
<th>Selected Best Practices for Rapid Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirements analysis,</td>
<td>change board,</td>
</tr>
<tr>
<td>buy vs. build planning,</td>
<td>daily build and smoke test,</td>
</tr>
<tr>
<td>readable high-quality code,</td>
<td>designing for change,</td>
</tr>
<tr>
<td>effective construction</td>
<td>evolutionary delivery,</td>
</tr>
<tr>
<td>practices,</td>
<td>evolutionary prototyping,</td>
</tr>
<tr>
<td>design (structured, object-oriented, etc.),</td>
<td>goal setting,</td>
</tr>
<tr>
<td>error-prone modules,</td>
<td>inspections,</td>
</tr>
<tr>
<td>automated estimating tools,</td>
<td>Joint Application Development (JAD),</td>
</tr>
<tr>
<td>integration strategies,</td>
<td>lifecycle model selection,</td>
</tr>
<tr>
<td>automated planning tools,</td>
<td>measurement,</td>
</tr>
<tr>
<td>automated scheduling tools,</td>
<td>outsourcing,</td>
</tr>
<tr>
<td>software configuration management (SCM),</td>
<td>principled negotiation,</td>
</tr>
<tr>
<td>staffing levels (how many and when to add),</td>
<td>productivity environments,</td>
</tr>
<tr>
<td>structured programming,</td>
<td></td>
</tr>
<tr>
<td>architecture,</td>
<td></td>
</tr>
<tr>
<td>effective planning,</td>
<td></td>
</tr>
<tr>
<td>quality assurance,</td>
<td></td>
</tr>
<tr>
<td>efficient meetings,</td>
<td></td>
</tr>
<tr>
<td>object technology,</td>
<td></td>
</tr>
<tr>
<td>parallel development,</td>
<td></td>
</tr>
<tr>
<td>reviews,</td>
<td></td>
</tr>
<tr>
<td>walk-throughs,</td>
<td></td>
</tr>
<tr>
<td>code reading,</td>
<td></td>
</tr>
<tr>
<td>leadership,</td>
<td></td>
</tr>
<tr>
<td>hiring top talent,</td>
<td></td>
</tr>
<tr>
<td>source code control,</td>
<td></td>
</tr>
<tr>
<td>testing, and</td>
<td></td>
</tr>
<tr>
<td>tracking.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. The Airlie Software Council’s “Project Caveats” of Worst Practices (1995)

1. Don’t use schedule compression to justify usage of new technology on any time-critical project.
2. Don’t specify implementation technology in the request for proposals.
3. Don’t advocate use of unproved “silver-bullet” approaches.
4. Don’t expect to recover from any substantial schedule slip (10% or more) without making more than corresponding reductions in functionality to be delivered.
5. Don’t put items out of project control on the critical path.
6. Don’t expect to achieve large positive improvements (10% or more) over past observed performance.
7. Don’t bury all project complexity in the software (as opposed to the hardware).
8. Don’t conduct the critical system engineering tasks (particularly the hardware/software partitioning) without sufficient software expertise.
9. Don’t believe that formal reviews provide an accurate picture of the project. Expect usefulness of formal reviews to be inversely proportional to the number of people attending beyond five.
**Process improvement**

Effective teams tend to have something else in common: they learn from their mistakes, improve performance over time, and maintain a high capacity to continue being effective in the future. A team's ability to do this depends on three other conduct factors: the team's awareness of its processes, its flexibility, and its capability for remembering what it has done in the past. All three are important.

**Process awareness**

Effective teams pay periodical attention to both their operational and teamwork processes. Hanson [1972] explains the importance of process awareness for team members:

"In all human interactions there are two major ingredients—content and process. The first deals with the subject matter or the task upon which the group is working. In most interactions, the focus of attention of all persons is on the content. The second ingredient, process, is concerned with what is happening between and to group members while the group is working. ... In most interactions, very little attention is paid to process, even when it is the major cause of ineffective group action. Sensitivity to group process will better enable one to diagnose group problems early and deal with them more effectively. Since these processes are present in all groups, awareness of them will enhance a person's worth to a group and enable him to be a more effective group participant."

The main type of processes Hanson is referring to are the teamwork processes within a group. Teamwork processes influence such items as morale, group dynamics, participation, styles of influence, leadership struggles, conflict, competition and cooperation. Additionally, software teams have to be aware of their operational processes, the strategies and methods used for structuring and tackling the content. Without it, software process improvement would be impossible; many software activities assume that developers are able to identify their operational processes (whether they be mature or ad-hoc) before they can attempt to
improve them. Fortunately, because software engineering traditionally focuses on development methodologies, tools and techniques, this second type of process awareness is usually less of a problem for software teams.

**Flexibility**

Effective teams are flexible. In studying how over a dozen field and laboratory groups completed various specified tasks in an allotted period of time ranging from one hour to six months, Gersick [1988] confirmed that groups do not always develop in a universal sequence of stages, but that the timing of when groups form and change the way they work is highly consistent. Each team in the study established in its first meeting a method of performing its given task and maintained the agreed-upon method until exactly half of the allotted time was used up. At this halfway point, without exception, all teams decided to pursue a different strategy to complete their task, and maintained this more efficient and effective approach until the task was completed. These consistent patterns of behaviour observed across the teams studied are described in Gersick’s time and transition model (also known as the punctuated-equilibrium model) [Gersick, 1988; Salas et al., 1992, pp. 7-8; Robbins, 1993, p. 289]. Using Gersick’s model, we can predict that most software teams will exhibit long periods of structural inertia interspersed with brief revolutionary changes triggered primarily by their members’ awareness of deadlines. As well, because this model suggests that all teams will undergo some form of significant change partway through their lifecycle, superior software teams can plan ahead for the time needed to update and modify project plans and other documents that may need to be amended when these transitions occur.

Thus, effective teams need to modify their processes partway through a project in order to remain effective even if their originally selected processes were optimal in the project’s early stages. However, research indicates only that inflexible teams unable to adapt to change are likely to be ineffective; the reverse is not necessarily true: being extra flexible has not been shown to further increase effectiveness. Excessive flexibility may actually impede performance by immobilizing a team with constantly changing goals, methods, structure or indecision.
Finally, effective teams establish some form of organizational memory that can survive the loss of a team member. Usually this involves writing things down: what was done, what happened, how were decisions reached, what worked, what did not work, and what should be done differently next time.

Awareness, flexibility, and organizational memory – how do these conduct factors correspond to CMM-style process improvement? Emphasis on defining processes to reach level 3 gives a team the capability to remember what it has done in the past; emphasis on measuring performance factors to get to level 4 helps make a team more aware of its process effectiveness; and organizations that reach level 5 will have flexibility ingrained in the culture. However, since most organizations are only at level 1 or 2 and very few ever reach level 5, effective teams may benefit from using the model more as a guideline while working on developing all three capabilities in parallel rather than one after the other. As well, teams should be aware that the model focuses mostly on improving an organization’s operational processes, and for a team to become and stay effective members need to also pay attention to improving their teamwork processes.

2.3 Summary of skills for software team leaders

The factors described in the previous sections are all correlated to some degree with team performance, satisfaction and future capabilities. The relationship does not unconditionally promise that all teams meeting these “criteria for success” will indeed be successful, nor guarantee that successful teams will have necessarily met all these conditions. Nevertheless, these internal and external factors have all been reported to influence team effectiveness, and Table 4 summarizes them into a set of practical guidelines for building an effective team. Understanding these factors would be beneficial for anyone leading software teams. Chapter 3 will build upon this knowledge to identify specific team skills that help all software professionals satisfy these criteria for success.
**Table 4. Summary of the factors contributing to team effectiveness**

<table>
<thead>
<tr>
<th>Life Cycle:</th>
<th>A team's present performance depends partly on how the team has matured over time:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Did it <em>form</em> effectively (get acquainted; set goals; establish a positive working environment)?</td>
</tr>
<tr>
<td></td>
<td>Does it <em>storm</em> effectively (manage conflict; surface issues; legitimize concerns; invite input and feedback; expect and accept tension)?</td>
</tr>
<tr>
<td></td>
<td>Does it <em>norm</em> effectively (develop cohesion and high-performance norms)?</td>
</tr>
<tr>
<td></td>
<td>Does it <em>perform</em> effectively (implement task and maintenance functions; interact with a high degree of high-level questioning; minimize dysfunctional self-oriented behaviour)?</td>
</tr>
<tr>
<td></td>
<td>Does it <em>adjourn</em> effectively (do members bring tasks to closure and maintain a capability to work together again in the future)?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Composition:</th>
<th>Effective teams have:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>adequate collective member competencies (skills, knowledge, interests, personalities,...)</td>
</tr>
<tr>
<td></td>
<td>uniformly heterogeneous membership</td>
</tr>
<tr>
<td></td>
<td>an equitable status hierarchy</td>
</tr>
<tr>
<td></td>
<td>a team size appropriate to the task’s complexity and its conjunctive/disjunctive nature</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenge:</th>
<th>Effective teams are challenged by:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a clear and common purpose</td>
</tr>
<tr>
<td></td>
<td>goals that are achievable, realistic &amp; motivating for their particular composition and context</td>
</tr>
<tr>
<td></td>
<td>tasks that are sufficiently interdependent for the members’ interaction skill levels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context:</th>
<th>The external environment and conditions imposed on the team facilitate task completion:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>team is given adequate authority/power/resources to make and implement its own decisions</td>
</tr>
<tr>
<td></td>
<td>evaluation and reward structures support goals, provide regular feedback, enhance performance and motivation, and identify individual member outputs.</td>
</tr>
<tr>
<td></td>
<td>physical work setting supports proximity &amp; privacy, culture holds high-performance norms, stress level is appropriate (not too high, nor too low), team has access to necessary tools, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cohesiveness:</th>
<th>An effective team with high-performance norms establishes cohesion:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>members are closely bound to team by mutual attraction, interdependency, and shared beliefs</td>
</tr>
<tr>
<td></td>
<td>team develops a history of success; team spends time together; members have many opportunities for interaction; strong team identity is present; interaction is fun</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conduct:</th>
<th>Effective teams interact and perform their tasks in ways that:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>are based on effective operational processes or best practices (so tasks are completed)</td>
</tr>
<tr>
<td></td>
<td>are based on effective teamwork processes (so the human resources are used to their potential)</td>
</tr>
<tr>
<td></td>
<td>facilitate process improvement: team pays periodical attention to both operational and teamwork processes, team is flexible and adapts to change, and maintains a persistent form of organizational memory.</td>
</tr>
</tbody>
</table>
What skills are important for software team members? The previous chapter identified factors correlated with effective performance and suggested that team leaders will benefit from being aware of them when making day-to-day decisions affecting the team. Now we want to identify the skills that benefit the other members of software teams. This chapter presents an extensive list of skills needed for software development. The list includes skills that are software-specific as well as ones that are more general-purpose, important for any task-oriented team. All of the listed skills may be useful for programmers working alone, but many (the subset of team skills) become particularly important when an individual joins a group. The most important general-purpose team skills for software engineers are described in detail because they are the most often overlooked skills in an industry that typically rewards only software-specific personal skills.

3.1 Candidate skills

Important skills for software teams obviously include the core computer science and engineering skills currently taught to undergraduates, and skills identified in the software engineering and computer science education literature. However, software teams also need some general-purpose skills for communication, teamwork, time and task management, learning, problem solving, leadership, entrepreneurship, and creativity. These types of general-purpose skills are often omitted or overlooked in software-specific skills lists.
To further confuse the issue, there is no consensus in the literature on how to label and categorize the many general-purpose "team" skills that have been identified. Similar skills have independently been classified under different labels; specific skills relating to communication, teamwork and dealing with people have been classified as foundation employability skills [Corporate Council on Education, 1993], enterprise skills [Gibbs, 1995a, p.66], transferable skills [Gibbs et al., 1994; Barrett & Cohen, 1996, p. 18], soft skills [Moss & Tilly, 1995], team skills [Losoncy, 1997], meeting skills [Bradford, 1976; Haynes, 1988], management skills [Whetten & Cameron, 1991; Charney, 1994], group work skills and co-operative learning skills [Gibbs, 1995a, p.66]. There also seems to be no rationale for why some skills are present in some lists and absent from others. Still other skills that are universally considered important (such as oral communication and leadership skills) are not universally defined, and so there is no agreement as to what the skill actually entails, nor of what behaviours demonstrate competence in the skill.

Because of these shortcomings of the literature, I have chosen to group skills into four categories: general-purpose personal skills, general-purpose team skills, software-specific personal skills, and software-specific team skills. As the concentric circles in Figure 5 illustrate, some categories of skills encompass others. I have defined general-purpose skills as skills that would be useful to individuals of any task-oriented team, whereas software-specific skills additionally include skills unique to the needs of software development. The set of personal skills includes all the skills necessary for software engineers working alone on a software system, while the set of team skills additionally incorporates skills that become important in a team environment to help use the team's human resources to their fullest potential.

The skills identified in Figure 5 include what I consider to be a representative sample of core software team skills, culled from over 200 candidate skills in various lists. This list should be treated as a dynamic, working taxonomy of skills. It serves as a framework for identifying broad abilities independently of their component skills, and for classifying and prioritizing more specific software skills as they are identified.
collaborating on technical documentation, using source code control, using make-files, following style guides, defining common interfaces, integrating code, critiquing technical ideas, communicating with developers and non-developers

making meetings efficient and productive, balancing workloads, estimating effort, tracking progress, team building (forming, norming), managing conflicts, integrative problem solving, making group decisions, process awareness & improvement, communication & interaction skills, managing customer expectations, understanding the product's user, effective use of power & influence, listening & questioning skills, giving effective oral presentations

managing stress, creative problem solving, self awareness, critical thinking, ability to work independently, time management, goal setting, numeracy, reading & writing, ethics & professionalism

programming in C/C++, data structures, algorithms, complexity analysis, network communication, client-server architectures, database design, graphics & user interfaces, specification techniques, basic hardware design, developing test cases

Figure 5. Important skills for software team members
What makes these particular skills important for software development? Some of the skills give team members the capability to make their teams effective; such skills are related to forming, storming, and norming effectively, motivating members to complete their tasks, and generally satisfying the characteristics of successful teams (as described in Chapter 2). Other skills help members avoid failure; for example, meeting skills are important, not because regular meetings guarantee success, but because never meeting almost always leads to poor communication and integration problems. Some skills (i.e. programming, developing requirements, design, integration, reviewing code, testing, debugging, maintenance, and documentation) are obviously necessary for completing typical software tasks effectively. However, to develop into an effective software team, members’ needs must be satisfied and the team’s future capabilities must be maintained throughout the process of completing the team’s tasks; for this reason, skills for effective project management, process improvement, conflict resolution, and building cohesiveness are equally important.

3.2 General-purpose team skills for software development

The rest of this chapter looks more closely at the most important non-software-specific team skills for software engineers. These skills improve the effectiveness of software teams, but are relatively unfamiliar to many software professionals and are often neglected in undergraduate software engineering programs, which traditionally teach only software-specific personal skills.

3.2.1 Project management skills

The number one software development problem plaguing software teams, according to a recent study of British Columbia software companies [Toth, 1997], is schedule overruns. Furthermore, according to a software manager interviewed, the few teams who do appear to have their schedules under control may be accomplishing this not through strong project management, but by purposely failing to mention specific deadlines in their contracts, enabling them to change delivery dates as impossible deadlines approach so that nothing is ever “late.” Software teams need more skills than this.
Project management skills help teams deal with many of the problems they encounter related to time pressures and deadlines. Such skills include:

- workload balancing (which requires skills for estimating effort, scheduling, and allocating roles and responsibilities dynamically),
- risk management, and
- basic progress tracking.

These skills are not just important for managers; teams and individuals benefit when all members have project management skills. There are at least three reasons for this:

1. The current trend in the high-technology industry is to promote technical people into management and team leadership positions because of the technical complexities of development. It is unrealistic to expect software engineers to master the elements of project management overnight when they move into this leadership role, and so developing project management skills and experience prior to promotion ought to be part of every team member’s job.

2. The effort and time required for software development is highly variable and dependent on many intangibles, such as the experience and skills of team members, on the suitability of the programming language, platform, tools and development paradigm used, on the quality of documentation and software components previously developed, on the complexity of the design, and so on. Even where management has technical expertise, estimates can often be radically improved by involving developers and others who, because they are closer to the task, have the potential to more accurately determine risks, estimates, and progress.

3. Team members with project management skills will feel a greater sense of control over factors affecting their work and will experience greater satisfaction. Often deadlines are proposed by others—management, marketing, customers, testers—as in: “We need this done by mid-July.” With poor estimation and risk management skills, team members have no clue if the request is reasonable until a deadline is imminent. Would it not be better if team members could know at the outset how...
much effort and time tasks should take so they could help control the schedule rather than have the schedule control them? At the least, greater control and involvement leads to higher levels of motivation and commitment to the project plan.

Kemerer [1997] has put together a comprehensive collection of papers relating to project management for software development. The readings and cases include surveys and in-depth discussions of estimation models, risk management, project lifecycles, prototyping, process tools (including CASE and OO technology), the management of software reuse and maintenance, and software process improvement. Additionally, readers interested in learning specific techniques and skills for better estimation, scheduling and progress tracking are encouraged to refer to the latest project management publications (e.g. Bennatan [1995]'s practices and techniques specifically for managing software projects).

3.2.2 Meeting skills

Meetings (both formal and informal) are often the primary place where team members interact, share information, solve problems, make decisions, and generate ideas. This is reflected in the often inordinately large amount of time spent by software professionals in meetings each week. Software teams benefit from making their meetings more efficient and productive.

Fortunately, the literature contains numerous strategies for improving meetings. Some of the simplest strategies to implement have the greatest potential for improving meeting efficiency and productivity when carried out at every meeting. I have selected seven of the most commonly cited general-purpose strategies for improving meetings and called them the "Meeting Basics" (presented in Table 5). The literature strongly correlates all of these strategies with more efficient and productive business meetings [Bradford, 1976; Charney, 1994; Gibbs, 1994a, 1994b; Harrington-Mackin, 1994; Harvey & Drolet, 1994; Haynes, 1988; Howell, 1995; Scholtes, 1998]. These strategies help members structure their content and participants to achieve the meeting's purpose and will improve meeting effectiveness when accompanied by adequate preparation and interaction skills.
Table 5. The Meeting Basics: Strategies for efficient and productive meetings

1. **Objective:** determine why the meeting is necessary (i.e. to distribute information, generate ideas, make a decision, or just build cohesion and have fun?) and then identify one concrete, relevant, attainable, task-oriented goal for it; ensure everyone understands the goal; define it in such a way that the group can recognize when it has been successfully reached so the meeting can be adjourned.

2. **Agenda:** have one, distribute it in advance, follow it, order items by urgency, and allocate time for items according to their importance and relevance to the meeting’s objective.

3. **Minutes:** record all ideas and decisions of the meeting, preferably in a manner visible to the group (such as on a flip-chart to eliminate errors, omissions and ambiguities right away); ensure that the minute-taker can distribute copies of the minutes to all participants soon after the meeting (within 24 hours if possible); identify action items clearly (what will be done, who is responsible, by when).

4. **Materials:** be prepared; at minimum the group should have writing materials (e.g. coloured pens and something persistent to write on), a calendar, a copy of the project schedule, a calculator, and minutes of the past meeting(s) (to review action items, risks, previous ideas and decisions made).

5. **Facilitator:** designate someone responsible for ensuring that all these Meeting Basics are met, as well as for paying attention to process issues (such as consensus, participation, seating arrangements), for keeping discussion focused on task, and for starting and ending the meeting (and individual agenda items) on time. The facilitator may delegate some of these tasks to others (e.g. to a time-keeper or a minute-taker).

6. **Next Meeting:** discuss the next meeting’s date, time, location, objective, and facilitator before adjourning and draft an agenda, clearly identifying who and what may be needed for the meeting.

7. **Self-Evaluation:** allocate time at the end of every meeting to formally reflect on the group’s process; establish an atmosphere and format for the facilitator to receive/give feedback (oral or written), and for all participants to comment specifically on what went well and what to improve next time.
The advantages of implementing these strategies are well documented. For example, having a single objective helps mobilize the group, helps keep the meeting focused on task, and improves motivation and member satisfaction. Failure to establish a single objective can have negative consequences for team effectiveness: “Meetings that go on and on with no visible product bring morale down quickly. Likewise, meetings with ten major concrete product goals paralyze those attending” [Williams, 1993, p. 86]. Similarly, discussing the next meeting can often take as little as sixty seconds of meeting time, but can spare members enormous hassle trying to schedule a mutually convenient time, date, and location once they have gone their separate ways. As well, it ensures that another meeting will indeed take place and can help in anticipating what information and people will be needed so the team can ensure they are there.

Unfortunately, without all team members having the experience and skills necessary for implementing and understanding the Meeting Basics, a leader’s attempt to make software meetings more efficient and productive may meet with enormous resistance. In a conversation with a junior programmer in a large MIS group, I received the following skepticism about taking time at the end of each meeting for self-evaluation:

“I’m sure that’s a good idea in theory, but it would never happen where I work. Our meetings are so long, sometimes up to three hours long! There’s no way we would stay for an extra five minutes at the end to do something like evaluate them. Besides, nothing ever takes just five minutes... we could be there forever arguing about how our meetings could be improved.”

This comment illustrates well the absence of both meeting skills and continuous process improvement in this software team. This all-too-typical group will ironically spend hours in unenjoyable, inefficient meetings because they do not recognize the value of spending just five minutes exploring ways to save potentially hours of time in the future by making their meetings more enjoyable, efficient, and productive. Furthermore, the quote suggests that this team does not understand the concept of having an agenda, of allocating time for each item, and of assigning a facilitator or time-keeper to ensure that time limits are
They would not have to stay an “extra” five minutes nor fear being there “forever” if they had the skills of managing their meeting time.

Some software teams may be so lacking in meeting skills that they never meet, or meet only sporadically. While face-to-face meetings are not mandatory for a software team to be effective, regular meetings do enable members to spend time together and build cohesiveness, may help members detect problems early by surfacing issues and concerns, provide a tremendous opportunity for all members to focus on an issue simultaneously and resolve it at once, and may improve productivity by routinely bringing tasks to the forefront of members’ attention. If face-to-face meetings are not possible, teams may experiment with holding virtual meetings in an Internet chat format, or try using audio or video conferencing software. The key element distinguishing meetings from other forms of group interaction is that meetings can provide synchronous communication opportunities in which all members may simultaneously partake.

Finally, members should understand that the Meeting Basics are not a panacea for all meeting problems. Meeting skills help teams structure their interaction to be more efficient, but will only help teams achieve the meeting’s purpose when members also have adequate information, knowledge, experience, opinions, ideas, attitudes and expectations to handle the content of the meeting and complete the team’s tasks, as well as basic interaction skills for working together effectively while processing the meeting’s content (discussed in Chapter 3.2.4).

### 3.2.3 Asynchronous communication skills

Outside meetings, software teams often communicate asynchronously. The term “asynchronous” designates communication that can occur at a different time for each participant, with expected delays ranging from a few minutes to an indefinitely long period of time [Edwards & Mynatt, 1997]. Telecommunications networks and the Internet have made the use of e-mail, newsgroups, bulletin boards, voice mail, and other forms of electronic asynchronous communication a viable alternative to face-to-face interaction for software development groups. Because such communication does not require team
members to be simultaneously logged on to their systems, nor to be at the same place at the same time, teams with access to e-mail (virtually every software team) will have tremendously greater opportunities for interaction than those without. As well, such modes of communication often provide teams with organizational memory (through the ability to store and later retrieve messages) which can often be processed by computers (searched, edited, rearranged and relocated easily) [Page, 1997, p. 11]. In fact, the use of e-mail is so common in the software industry today that its benefits can safely be taken for granted until one has to communicate with someone outside the industry.

However, teams in general have not always had access to this technology, and the sociological transition from depending almost entirely on meeting face-to-face to expecting every team to have asynchronous communication options has occurred relatively recently. Therefore, little research is available on what skills and processes are needed in teams to make electronic asynchronous communication work most effectively. Nevertheless, team members clearly need new skills beyond knowing how to send an e-mail message; effective social practices for synchronous interaction do not translate directly into the best practices for asynchronous collaboration (and many people apparently do not even apply these simple standards of social practice in the new impersonal electronic medium [Cranor, 1996]).

As a start, I suggest team members learn the asynchronous group communication skills described in Table 6. These skills and guidelines should help ensure that e-mail and other similar forms of communication remain efficient and effective. Continued research in the areas of Computer Supported Co-operative Work (CSCW) and TeleLearning may in the next few years add insight into additional skills and strategies for software teams that cannot or choose not to meet face-to-face during a project.
Table 6. Guidelines for effective electronic asynchronous group communication

1. **Provide context with each message.** A “yes”, “no” or “I agree” on its own can be very ambiguous, especially when messages are retrieved in an order different from what the sender expected.

2. **Structure each message so that recipients are not overwhelmed by too much content.** For example, provide an executive summary at the top of a long message, number questions, emphasize and organize important points with a suitable indentation or numbering scheme, remove irrelevant parts of quoted text, and so on.

3. **Organize multi-message communications so members are not overwhelmed by too many messages.** When a new topic to be debated is introduced, or in deciding when to schedule a meeting, it can sometimes be more effective to have members respond to the original poster rather than to the entire group, so long as the original poster collects all the responses into a single message (or at least prepares a summary) and sends that back to the entire group afterward. This requires that the original poster be skilled in anticipating the type of response that a posting will get so that appropriate instructions can be included with the original message, such as: “reply to me by Friday, October 14th, and I’ll then summarize the responses for everyone.” The importance of this becomes very apparent when a vote or group decision must be made asynchronously.

4. **Do not send messages to the entire group that are relevant only to an individual.** Irrelevant postings can quickly clog people’s inboxes, reduce the perceived significance of subsequent messages, and result in less attention being paid to important communications.

5. **Establish a policy of open communication between all group members.** When communication does occur privately about a group matter, inform the entire group of the discussion’s outcome (“being careful to say ‘we recommend’ rather than ‘we decided’ unless the group has given them the authority to make the decision” [Cranor, 1996]). Similarly, members should keep everyone informed of progress they have made in carrying out their responsibilities and of difficulties they may be having completing a task they volunteered to do (due to lack of time, lack of expertise, or any other reason). The designated leader should communicate often with all other members and could privately collect reports to summarize for the group.

6. **Establish a policy on how regularly people should check for messages and determine a reasonable expected delay between communication.** Otherwise, people will not know if a lack of response is “because they haven’t been checking their email, they don’t really care about the issue in question, they don’t have enough information to make up their minds, or they are opposed to the proposal and are hoping a quorum is never reached so that the issue won’t get decided.” [Cranor, 1996, p. 17].

7. **Establish a “no flame” policy** [Cranor, 1996, p.17]. Conflict is destructive in the form of personal evaluative attacks intended to hurt rather than help. Unfortunately, flame wars can easily erupt in an impersonal asynchronous medium. Feel free to offer constructive criticism and feedback about ideas and processes, but avoid the trap of criticizing people or of misinterpreting someone’s written comments as a personal criticism. Realize that even if someone writes something in a stupid or insensitive way, it does not mean that they are a stupid or insensitive person, and it is not constructive to suggest that they are.

8. **Ensure that someone is responsible for maintaining the communication system.** This person might create suitable mailing lists or conference areas, maintain members’ contact information, keep a message archive, and ensure that communication is not interrupted for an avoidable reason (such as missed account payments).
3.2.4 Interaction skills

During interaction, there are many different things that members can say, and even more ways to say them. Each comment can have either a functional or dysfunctional effect on the task or team dynamics, regardless of whether or not members are aware of it. Interaction skills give team members the tools they need to consciously manipulate what they say and how they say it to achieve some desired outcome. Skilled members are aware of the interaction process, and can anticipate what reaction they want and can expect from the different kinds of utterances they might make in each situation.

What kinds of utterances can people choose to make? At the most basic level, an utterance can be either a statement or a question. According to Harvey and Drolet [1994], productive teams focus more on asking questions; high levels of question-asking and listening can apparently shorten decision time by 30% and is more empathetic, inclusive and efficient. People can learn to distinguish between asking open and closed questions to elicit different types of information, and between using commands, questions, hints and prompts to exert a level of assertiveness appropriate to the urgency of a situation. Unfortunately, asking good questions is difficult to learn since most people prefer to tell, and typically only 5% of utterances are questions or answers to questions. Haynes [1988, pp. 41-45] provides useful guidelines for formulating questions to stimulate discussion during a meeting.

At a higher level of analysis, Bradford [1976] and other researchers have identified numerous distinct functional roles that people's statements and questions can fulfill. Their lists provide an excellent guide to the kinds of interaction skills that would benefit members of software teams, and some of these roles of group communication are summarized in Table 7. (My descriptions are adapted from Bradford [1976, Chapters 5-6]; Tubbs [1988, pp. 229-233]; and Marcic [1990, pp. 380-381]).
Table 7. Functional roles of communication in group interaction

<table>
<thead>
<tr>
<th>Functional Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initiating</td>
<td>Involves proposing a group task activity or goal, defining a problem, or suggesting a new idea or way to solve a problem so that the group can get started on a task and move from ideas to action.</td>
</tr>
<tr>
<td>2. Information and opinion giving</td>
<td>Involves contributing facts, data, or personal opinions to help prevent narrow distorted decision making.</td>
</tr>
<tr>
<td>3. Information and opinion seeking</td>
<td>Involves asking for relevant information, ideas, suggestions or opinions to encourage hesitant members to speak and to prevent premature closure of issues, groupthink, and the unnecessary withholding of information.</td>
</tr>
<tr>
<td>4. Coordinating and summarizing</td>
<td>Involves pulling together related ideas, restating suggestions, and offering a decision or conclusion for acceptance or rejection in order to generate new creative solutions, to prevent a group from becoming polarized into two argumentative sections, or to mobilize a group to move on.</td>
</tr>
<tr>
<td>5. Clarifying and elaborating</td>
<td>Involves defining terms, clearing up confusions, identifying issues and alternatives, and listening for half-stated contributions to question if they need elaboration in order to prevent potentially good ideas from becoming lost when half-stated, unheard, ignored or misunderstood, and (even if the point turns out not to be useful) to ensure everyone feels part of the group and remains active and involved.</td>
</tr>
<tr>
<td>6. Evaluating</td>
<td>Involves critiquing ideas (as opposed to behaviours) in order to identify problems related to the team’s task or to assess the options involved in a decision.</td>
</tr>
<tr>
<td>7. Consensus testing and terminating</td>
<td>Moves the group towards decision or action; involves asking to see whether the group is nearing a decision, sending up a trial balloon to test a possible conclusion in order to bring the group closer to a decision, and checking for consensus, agreement or disagreement.</td>
</tr>
<tr>
<td>8. Structuring and planning</td>
<td>Involves developing agendas, listing things to do, setting priorities, allocating time for discussion topics, structuring and planning the team’s direction, and preventing topic-jumping or going off on tangents in order to keep the group on target.</td>
</tr>
<tr>
<td>9. Energizing</td>
<td>Involves suggesting a short break, changing the pace, or using humour, novel ideas or some statement of feeling to help a self-activated group generate energy and maintain momentum.</td>
</tr>
<tr>
<td>10. Gatekeeping</td>
<td>Involves suggesting procedures for sharing information and bringing into discussion (or cutting off and interrupting) people to keep communication open and facilitate participation of all members.</td>
</tr>
<tr>
<td>11. Encouraging</td>
<td>Involves being friendly and responsive to others, offering praise, accepting others’ points of view, and being open-minded in order to ensure that ideas are not prematurely discarded, make everyone feel they are a valuable part of the team, and ensure all members remain active, involved and satisfied.</td>
</tr>
<tr>
<td>12. Harmonizing and consensus seeking</td>
<td>Involves mediating a conflict by seeking common ground, acknowledging emotions, and locating solutions agreeable to all.</td>
</tr>
<tr>
<td>13. Compromising</td>
<td>Involves displaying a willingness to compromise or admit error when one’s own idea is involved in conflict, to maintain group cohesion and enable the group to find a solution acceptable to all.</td>
</tr>
<tr>
<td>14. Giving and receiving feedback</td>
<td>Involves soliciting and giving direct, specific, descriptive, constructive, immediate and non-evaluative feedback about people’s behaviours without critiquing personalities. It also involves listening openly and non-defensively to feedback.</td>
</tr>
<tr>
<td>15. Processing and Standard setting</td>
<td>Involves reviewing how the group is performing its tasks so that it can improve; may also involve testing whether the group is satisfied with its procedures by pointing out explicit or implicit norms in order to test the group’s commitment to efficiency, fairness, and open communication.</td>
</tr>
</tbody>
</table>
The important thing for software team members is not to memorize this list, but to become aware of the wide variety of contributions they can make during group interaction and learn to anticipate what effect their contributions will have on the team so they can move more efficiently closer to its goals. When they have these interaction skills they will, for example, be able to anticipate if evaluating a suggestion will cut off further idea generation, and they can use this awareness to decide whether or not that would be productive for the team at that particular moment in time. Members will be better able to balance the voicing of opinions against the need to seek new data, or summarize, coordinate and clarify previous suggestions. Hesitant members may become more involved when they realize they do not need to voice an opinion or present new information to contribute effectively in a facilitative role. As well, members will be better able to maintain team cohesiveness when they can avoid saying things in ways that offend others. Some companies, including Anderson Consulting, are already teaching their software professionals to be more aware of how they interact; they videotape team problem-solving sessions and have members analyze the interaction afterward to identify individual strengths and weaknesses in fulfilling the same task and maintenance-related roles of communication described here [Anderson Consulting, 1994]. A sample worksheet for such an activity is included in the Appendix.

3.2.5 Conflict management skills

Conflicting opinions and expectations are inevitable between software developers, testers, managers, marketers, customers, users, and other concerned parties involved in software development. People may disagree about schedules and estimates, the importance of certain features, the best design for a software component, and how to make trade-offs between performance, usability, functionality, and maintainability. Sometimes even seemingly trivial issues, like how to name a particular variable, can turn a software team into a war zone.

Why is conflict so prevalent and likely in a software team? Many of the antecedent conditions for conflict [Whetton & Cameron, 1991, pp.397-399] are common in this environment: high levels of interdependency; differences in values, power, status, objectives and expectations; role ambiguity; scarce
resources (especially time); rapid change and uncertainty; a history of previous conflict; high stress; and a highly competitive organizational culture and group norms. These factors are evidenced by the fact that software developers are often working in unfamiliar territory, trying to design innovative products with new technology on a tight schedule, where the best approach is unclear, where the amount of effort required may be impossible to predict, and where mistakes and delays can cost a company its market-share and the team’s jobs.

What skills do software team members need to resolve conflicts effectively? First and foremost, they need to know how to resolve issues by consensus. Of all the ways of resolving conflicts (shown in Figure 6), this method is most difficult; collaborating requires high levels of assertiveness and cooperation from all members involved, who will need to use the interaction skills identified in Chapter 3.2.4 and apply to the conflict situation an effective group problem-solving process (described in Chapter 3.2.6). However, software teams have no alternative. Ignoring, suppressing, or “smoothing over” the conflict is an ineffective method of resolution. They often cannot afford to settle for compromise: meeting half-way when choosing a delivery date does not make sense—the corresponding effort required often cannot be compromised; meeting half-way between design alternatives is even worse—the resulting feature may be neither usable nor useful (as in “half a help system”). Similarly, members cannot afford the consequences of a non-unanimous vote, of forcing a preference on others, or of accommodating all the time: the entire team needs to feel satisfied by the resolution so that everyone buys into it, implements it, and continues working together for the rest of the project.

Boehm and Ross [1989] describe the implications of not seeking out day-to-day conflicts during software development and changing them into win-win situations; they propose not only that conflict management skills will help software teams, but also that “making everyone a winner” is central to effective software project management. They detail several skills necessary to create win-win situations, including: being able to determine how people want to win through identifying people’s underlying motivations, interests, concerns and criteria, being able to establish realistic expectations, and being able to search out or invent new options for mutual gain.
Unfortunately, one of the biggest hurdles to resolving conflicts collaboratively is that conflict often makes people emotionally uncomfortable, and so the natural preference is to avoid it in the first place. It is therefore very important that software team members intellectually appreciate the value of conflict, and understand that it is inevitable and not necessarily bad for a team. Without conflict, organizations generally fail in competitive environments: “Members are either so homogeneous that they are ill-equipped to adapt to changing environmental conditions, or so complacent that they see no need to improve the status quo. Conflict is the life-blood of vibrant, progressive, stimulating organizations. It sparks creativity, stimulates innovation, and encourages personal improvement” [Whetton & Cameron, 1991, p. 395], particularly when the conflict is over ideas or processes instead of incompatible personality preferences, and is resolved collaboratively. Additionally, software team members need experience with all the modes of managing conflict (avoiding, forcing, accommodating, compromising, and collaborating) so that they can be more flexible in the approach they choose, and can develop skills and confidence in handling the tense, emotionally charged environment typical of interpersonal confrontations.
Many books contain guidelines for handling confrontation and building consensus, and so specific techniques will not be discussed here (although integrative problem solving is discussed in the next section). Some publications that I have found useful and relevant include: *Getting to Yes* [Fisher & Ury, 1991], *More Than 50 Ways to Build Team Consensus* [Williams, 1993], *Developing Management Skills* [Whetten & Cameron, 1991], *The Dynamics of Bargaining Games* [Murnighan, 1991], *The 7 Habits of Highly Effective People* [Covey, 1990] and *How to Win Friends and Influence People* [Carnegie, 1981].

### 3.2.6 Group problem-solving and decision-making skills

Problem-solving skills are essential for becoming a proficient programmer or software engineer, and are therefore a typical strength of software team members. However, generating ideas, solving problems, and making decisions as a group requires additional skills. The problem and criteria for an acceptable solution need to be defined and communicated more precisely than when working alone. An effective problem-solving process needs to be made explicit and adhered to by all members, and members need to know and use techniques for listening, fostering creativity and avoiding groupthink.

Software teams benefit from group problem-solving and decision-making skills for a number of reasons:

1. Integrative problem solving is central to effective conflict management (as explained in 0).

2. Following a standard process will make the team more efficient, keeping people from rehashing ideas as the group is trying to make a decision, or from evaluating ideas too early in the process, and so on.

3. Group decision making can help in anticipating consequences of decisions and in generating acceptance for solutions. Even though a problem may be solved by one person working alone, others in the team will likely be impacted by the solution, especially given the highly interdependent nature of software development, and others may also encounter a similar problem at some other time over the course of the project. Having multiple people independently solve the same problem over again is
inefficient. Having multiple people collaborate in solving the same problem can potentially lead to more consistent, creative, hybrid solutions.

(4) Even though team members may recognize the value of meeting regularly, they may become frustrated if meetings are primarily informational in nature (especially in software teams where everyone has access to e-mail which can distribute information efficiently and often more effectively than a meeting). Group problem-solving and decision-making skills will enable teams to tackle brainstorming, problem-solving, and decision-making tasks in their meetings to make much better use of the collective expertise, energy, and time of team members.

(5) Even though team members may recognize the value of making group decisions, they may find themselves unproductive if every decision goes to the group. Group decision-making skills help members identify which decisions are most suitable for being made by the group, and which ones are better left to the individual. For example, minority decisions may be preferable when the issue relates to someone's immediate safety or when one person is the only acknowledged expert. Majority decisions may be preferable to consensus in a time crunch, if the issue is very divisive, or if there are too many people to negotiate a consensus and their commitment is not essential [Charney, 1994, 59].

So how do team members solve problems, generate creative ideas and make decisions effectively in a group? Understanding and being able to follow suitable processes for structuring these activities is key. Some procedures that members should be aware of and can choose between are listed in 0. A process for integrative problem solving is described in more detail in Table 9, and rules for brainstorming that facilitate the development of ideas are suggested in Table 10. Miller [1987] provides additional ideas for fostering innovation in task-oriented groups, and for avoiding groupthink.
Table 8. Procedures for teams

<table>
<thead>
<tr>
<th>Procedures for Solving Problems:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Integrative Problem Solving (see Table 9)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedures for Generating Alternatives:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Brainstorming (see Table 10)</td>
</tr>
<tr>
<td>• Buzz sessions (breaking into sub-groups)</td>
</tr>
<tr>
<td>• Brainwriting</td>
</tr>
<tr>
<td>• Fishbone Diagrams</td>
</tr>
<tr>
<td>• Affinity Diagrams</td>
</tr>
<tr>
<td>• Nominal Group Technique</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedures for Choosing among Alternatives:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Voting</td>
</tr>
<tr>
<td>• Consensus</td>
</tr>
<tr>
<td>• Multi-voting</td>
</tr>
<tr>
<td>• Nominal Group Technique</td>
</tr>
<tr>
<td>• Criteria Based Ranking</td>
</tr>
<tr>
<td>• Criteria Based Rating</td>
</tr>
<tr>
<td>• Criterion Based Paired Comparison</td>
</tr>
</tbody>
</table>

Table 9. Integrative problem solving process

1. Study/discuss/analyze the situation. Data is often preferable to people's opinions, especially for urgent, complex, contentious or emotional issues.
2. Define the problem.
3. Set an objective (the end result you want to accomplish), and state imperatives or desirables (what criteria your solution should satisfy).
5. Establish evaluation criteria. Seek out hidden agendas and concerns.
7. Choose among alternatives, using the authority of one person or of the whole group, as appropriate.
8. Plan to implement the decision (appoint someone to be responsible, ensure they have necessary authority, specify checkpoints and expectations), then implement the plan.
9. Evaluate results, and if they aren't as expected, determine cause (e.g. Faulty implementation? Wrong problem definition?).

(Steps 1-7 suggested by Haynes [1988, p. 32]. Steps 8-9 adapted from Miller [1987, pp. 136-137].)

Table 10. Rules of brainstorming

1. Visibly list all ideas offered by group members.
2. Do not evaluate or judge ideas at this time.
3. Do not discuss ideas except to clarify understanding (both criticism and discussion stifle the development of ideas and transfer energy away from generating ideas to defending them).
4. Let the sky be the limit. It is easier to eliminate ideas later than to accumulate them.
5. Repetition is okay. Do not waste time sorting out duplication.
6. Encourage quantity. The more ideas, the greater the likelihood of a useful one.
7. Do not be too anxious to end this phase. When a plateau is reached, let things rest and start again (better quality ideas can surface after allowing some time to think, or after leaving the list for a while and returning to it later).

(These rules suggested by Haynes [1988, p. 34].)
3.2.7 Team-building skills

Because a team’s effectiveness is closely tied to its stage of development, members benefit from knowing how and when to form, storm, norm, perform and adjourn effectively. Team-building skills will help members move through these stages of development.

Based on the team performance factors described in Chapter 2, important team-building skills include:

- Skills for getting acquainted (including being familiar with possible icebreakers to use, and ways of accepting a new member into the group).
- Skills for establishing a common purpose (including being able to identify goals and expectations and break these down into appropriate challenging tasks).
- Skills for allocating roles and responsibilities to satisfy both the skill demands of the tasks to be accomplished as well as the needs and interests of the members accomplishing them.
- Skills for storming effectively (including managing conflict, surfacing issues, legitimizing concerns, inviting input and feedback, and expecting and accepting tension).
- Skills for developing cohesion (including knowing to meet frequently, recognize accomplishments, build a history of success, establish a strong team identity, have fun together, and build trust and interdependence).
- Skills for recognizing and reinforcing high-performance norms.
- Skills for bringing tasks to closure

Finally, team members will benefit from understanding the development model and other factors related to effectiveness described in Chapter 2. Being aware of why they need to establish common goals and build cohesion is as important as knowing how to accomplish these tasks for them to proactively make their teams more effective.
3.2.8 Skills for understanding the user or customer

Most software teams would not exist without customers in need of their products or solutions. Knowing how to satisfy the user or customer’s needs and expectations involves skills that go beyond being able to meet the requirements of a contract. Requirements can be ambiguous, incomplete, inconsistent, and not grounded in reality, and so it is possible without skills for understanding the user to build a system that “meets spec” but is still a failure [Lewis & Rieman, 1995]. Successfully satisfying the user and customer requires an ability to communicate with non-technical people, humility (to recognize that you may not be the best-qualified person to make certain design decisions), and empathy (to put oneself in someone else’s shoes when making various design decisions and to appreciate why things may need to be done in a certain way). Additionally, knowledge of the application domain is valuable to everyone working on the project: teams developing software for the banking industry benefit from knowledge about finance, while teams developing air traffic control system software require knowledge about airways, aircraft types, flight plans, lateral and longitudinal separation, aviation hazards, and so on [Hughes Aircraft, 1993].

3.2.9 Process awareness

The previous chapter (2.2.6) explained the importance of process awareness for process improvement. Yet, improving team effectiveness is itself a form of process improvement, and awareness of teamwork process is essential for developing any of the team skills described so far.

Unfortunately, because operational subtasks typically receive more attention than teamwork subtasks, process improvement efforts usually focus more on the operational processes and skills used to complete tasks, rather than on the underlying fundamental teamwork processes and skills that determine if a team uses its human resources to the fullest potential. As well, being able to identify the processes that have developed naturally within a group is a difficult skill to learn. As Marilyn vos Savant [1996, p. 43] wrote: “We can learn the necessity to ‘question’ our assumptions, but, to do that, we first have to ‘recognize’ our assumptions—and that’s a lot harder. This is arguably the greatest logical weakness of bright people.”
3.2.10 Leadership skills

Leadership skills deserve mention because they are so frequently identified as important. However, as with team skills, leadership is not a well-defined concept. What behaviours can demonstrate leadership?

- Leadership is having someone initiate something when everyone is staring at the walls in a meeting.
- Leadership is having someone summarize, clarify, restate or coordinate many different ideas that have been put forward during discussion.
- Leadership is having someone record the ideas and decisions of a meeting so that none are lost.
- Leadership is having someone clarify the next meeting’s time and location before everyone leaves.
- Leadership is having someone periodically question the team’s assumptions, processes or direction.
- Leadership is having someone surface hidden agendas or concerns that may be causing conflict.
- Leadership is having someone organize a fun social event to help build cohesiveness.

These and many other behaviours demonstrate leadership. From talking with dozens of people during my research, I found that often when things go wrong, the designated leader or “lack of leadership” is blamed. Yet, most of these “leadership” skills are team skills that anyone can possess and use. The more team members who share them, the more likely a team will become and stay effective.

True, performing these functions may be the designated leader’s responsibility; however, a leader can only facilitate positive behaviour—it is ultimately the conduct of each and every member that determines whether a team will be effective or not. Furthermore, since the high-technology industry tends to promote technical people into official team leadership roles, it makes sense to develop these skills in all members as early as possible. Thus, leadership skills are really just another way of referring to the many general-purpose skills already identified in this chapter that benefit all members of software teams.

3.2.11 Other skills

The skills listed so far are particularly important for every member of a software team. While this is only a subset of the many skills that benefit teams, other skills are not necessary for all team members and so
were not given as high a priority, or are considered fundamental personal skills (as opposed to being team skills). Nevertheless, software teams will benefit greatly when at least one or more members have some of the following skills as well:

- **Oral presentation skills.** Team presentations are an important part of winning contracts, meeting customer expectations, marketing products, and communicating a team’s objectives or accomplishments to other groups in the company.

- **Interviewing skills.** In Chapter 2 we saw how significant are the competencies that members bring to a team; being able to identify and attract technically skilled people who complement the team’s heterogeneity and dynamics is very important.

- **Documentation skills.** This is a personal skill valuable for all team members, and definitely essential for those members who will be developing proposals, requirements and design documents, recording processes, commenting code, or producing user documentation; the quality of written documentation determines whether a team satisfies the customer, earns a good reputation, or is able to build on its past performance anytime in the future.

- **Facilitation skills.** In addition to having all members use the interaction skills identified earlier, teams benefit from having a member capable of stepping back from the content to take responsibility for facilitation of teamwork processes when necessary. As well, some software development techniques require a skilled facilitator to collaborate successfully with users (e.g. Joint Application Design and Participatory Design [Schuler & Namioka, 1993; Baecker et al., 1995, pp. 189-190]).

- **Creativity.** Fluency and flexibility of ideas and thought patterns is valuable for both individuals and teams. Whether it is a talent accessible only to a gifted few, or a skill that can be learned, creativity can be enhanced through many different techniques [Miller, 1987; Mountford, 1995].

- **Sponsorship.** This is not really a single skill in itself, but teams need someone to look out for their interests, secure the necessary resources, and see that they are suitably rewarded for their efforts.
Leading teams is decidedly not a matter of ‘You either got it, or you ain’t.’ Being an outstanding team leader is no more a natural talent than being an effective team member. Both roles require new skills and lots of practice.” [Zenger et al., 1994, p. 15]

As this quote from Zenger et al. reinforces (and as was illustrated by the competency model in Chapter 2.2.2), people are not born with team skills; team skills can and must be acquired over time. How can we guide the effective acquisition of team skills by software team members? One way is to make an effort to teach them in post-secondary software engineering, information technology and computer science programs. This chapter describes the special skills that trainers and educators need in order to teach team skills. Its recommendations are based on the literature, interviews, and teaching experience within an undergraduate software engineering project course (described in more detail in Chapter 5).

4.1 How should team skills be taught?

Some form of team skills training, assessment, and practice opportunities are typically found in commerce, education, counselling psychology, medicine, and other disciplines beyond computer science where graduates must work closely with other people on the job. Undoubtedly team skills can be taught effectively. However, the students who are attracted to those fields may arguably be more aware of the relevance of team skills to their careers than computer science students entering a field stereotyped by loners without social skills. Those other faculties may also attract a greater percentage of students who
already have fundamental prerequisite skills, experience, and people-oriented personalities that enable them to effectively develop their team skills throughout their degrees. Thus, just because those other faculties can teach team skills effectively does not necessarily mean the same strategies and techniques will transfer well to computer science.

My research supports several general principles on how to best teach team skills to software engineers. The primary educational objective should be to develop the general-purpose team skills identified in Chapter 3 in the context of software development. Courses should integrate practice opportunities, formal training, monitoring and assessment for each of the skills to be learned. They should ensure that team members are able to meet frequently and regularly, are given a suitably challenging assignment, and have sufficient shared resources to accomplish the team task. The learning environment should encourage risk-taking and accept mistakes. In introducing the less technical team skills, instructors should not fight the culture, but should instead work with it to minimize the cultural changes required for the software community to adopt and respect these skills. Finally, courses teaching team skills need to promote mastery at the expense of a potentially narrow spread in grades; our goal should be to develop a high level of competence in these skills in all software engineers, regardless of their technical abilities in other areas. These points are all described in more detail in the next few subsections.

4.1.1 Integrate practice, theory, monitoring and assessment

How are skills learned? Skill learning involves a cyclical process (shown in Figure 7) which may need to be repeated many times [Gibbs et al., 1994, pp. 13-17]. People can start anywhere in the cycle – with a new experience, with reflections on past experiences, with theory, or with planning to tackle a new situation. These four dimensions of learning are also described in Kolb’s experiential learning model which outlines the importance of concrete experience, reflective observation, abstract conceptualization, and active experimentation for skill acquisition [Svinicki & Dixon, 1987].
To facilitate effective skill acquisition and learning within the context of an academic course like CPSC 319, all four aspects of the process need to be supported. The arrows in the above diagram indicate what a course can provide to facilitate progression through the cycle. To help students “Do” requires practice opportunities in the course that demand use of the skill to be learned. To help students “Reflect” requires that students have ways of monitoring their effectiveness and get into the habit of reviewing their performance; either the course needs to give them regular feedback or students need to learn to give it to themselves by using checklists or some other means of self-assessment. To help students “Form Principles” requires formal training; “if students lack a skill then they cannot suddenly use it, and students often start from such a low base-line that they need help before they can even start” [Gibbs et al., 1994, p. 15]. To help students “Plan” requires that the course assess the skills, so that students take the skills seriously (otherwise students will strategically emphasize only what counts and is rewarded).
In applying the model to the teaching of team skills we see why it is important for computer science programs to include team projects: software engineering students need multiple experiences using team skills in the context of software development. However, we see that practice alone does not make perfect: courses also need to provide some formal training in using each of the component team skills, give feedback so students can learn what being effective is all about, and reflect in their grading schemes the importance of team skills in addition to technological skills.

4.1.2 Create a context to support effectiveness

Chapter 2 identified many variables that influence team success, some of which are beyond a team member’s control. In order for students to experience working in an effective team, the context factors in the learners’ environment must support team effectiveness.

To help teams become cohesive (and thereby provide members with an opportunity to practice skills of working in a more effective team), members need to have reason to meet frequently. The course workload needs to be fairly intense: spreading effort thinly and unevenly over two semesters is less desirable than channelling student energy into a compact and challenging project lifecycle of a few weeks or months. As well, to encourage students to cooperate and work interdependently the project itself needs to be large and complex, yet also easy to partition into autonomous subtasks that give every member something challenging and identifiable to do. To ensure that groups can find a common time to meet regularly, the algorithm used for allocating students to teams should pay some attention to members’ schedules.

Teams need a place to hold regular meetings. Access to private meeting rooms that can be booked in advance is desirable. The rooms should have chairs that can be moved around so that members can experiment with different room layouts and be able to sit facing one another. Tables are important, as well as walls on which to mount a whiteboard, blackboard, or large sheets of paper for writing on that everyone can see. In an ideal world, teams would also have access to the Internet and their shared files.
from this room so that unanticipated issues arising during the meeting can be resolved immediately with concrete data and facts, rather than people’s memories and opinions. Additionally, it would be useful to mount a video camera in the room, so that team members can record and observe themselves periodically to develop interaction skills. Finally, it would be useful if teams could leave materials on the walls and whiteboard between meetings, be encouraged to write on large sheets of flip chart paper that can be taken down after each meeting, or else have access to an electronic whiteboard whose contents can be saved between sessions; otherwise they will be forced to triplicate effort by copying ideas verbatim off the board at the end of the meeting and rewriting them onto the board for the next.

Teams require shared resources: disk space, print quota, photocopying funds (for distributing copies of instructor feedback), team binders (for storing a hard-copy of agendas, minutes, schedules, processes and other team artifacts), and meeting supplies (including coloured markers, flip chart paper, putty or tape for attaching things to walls, a calculator, a calendar, and special materials for generating ideas such as paper, scissors, index cards and sticky notes). While the responsibility for purchasing these shared materials can be left up to the individuals in each team, courses can support teams by providing some of these resources (even if they have to charge individuals a supplementary course fee for the “team toolkits”). Otherwise teams will run into a problem of ownership, where nobody wants to donate items or funds for shared course materials that unlike a textbook are not clearly theirs to own. Furthermore, individuals sometimes cannot provide the resources (for example, a team programming project may require more disk space than any individual team member is allocated, and often only a system administrator can set up permissions so that team members and instructors can share files without making them readable to the world).

4.1.3 Promote mastery

Course grading schemes can severely impact how well students learn team skills. As has already been mentioned, the grading scheme needs to recognize teamwork in addition to individual achievement so that students take team skills seriously. However, another issue plays an even bigger role: are course grades
expected to sort students (and achieve a normal distribution or “bell curve”), or are the grades expected to measure learning and encourage mastery of the course material?

If there is an implicit underlying assumption that a sufficiently rigorous course will necessarily have some proportion of students performing poorly, then mastery of team skills by all students in such a course may be an impossible goal.

“Groups tend to produce better work than individuals and so group work tends to produce higher average marks than individual work. Also the effect of putting mixed students together in groups is to make groups less varied than individuals – producing a narrower spread of marks between groups than between individuals. The common outcome is a high mean and a low standard deviation with no fails.” [Gibbs, 1995a, p. 19]. To achieve a normal distribution, instructors may have to artificially inflate or exaggerate minor discrepancies between students, or do excessive individual assessment (quizzes, etc.) which would encourage students to abandon much of their group work and focus energy on the things that differentiate them the most. Or alternatively, since students’ prior knowledge is more likely to be normally distributed, the desired result could be achieved by assessing skills that are not taught in the course, or are taught very poorly. Neither method facilitates the learning of team skills.

If we really want to teach software team skills, we have to de-emphasize the importance of differentiating students through grades and make it possible for all students to achieve mastery of the material. If there were only jobs available for the top 10% of computer science graduates, then it might be useful to sort students with grades. However, the software industry is in high need of qualified people and will have positions for nearly 100% of our graduates. Because an individual’s team skills affect the performance of the entire team, everyone suffers when we supply graduates without superior team skills. Just as we would not tolerate a system that grants driver’s licenses to people who know only 50% of the rules of the road, we should expect software engineering education to set a high standard of zero-tolerance for incompetence in general-purpose team skills. Instead of bemoaning a narrow spread with many well-deserved ‘A’s in a particular team skills course, departments should critically examine how any student
could have been allowed to pass without legitimately earning an ‘A’. This shift in perspective would at least ensure more effective software teams down the road.

4.1.4 Expect Imperfection

This may seem to contradict the previous recommendation to promote mastery, but in fact, accepting initial imperfection is pivotal to helping students master skills. Only if mistakes due to prior inexperience are forgiven by the marking scheme will students feel encouraged to take risks and try things beyond their comfort zones.

For example, when students first learn project management skills, they may be asked to develop a project plan with estimates of how long tasks will take. However, instead of grading them on the accuracy of those initial estimates (which could be based largely on guesswork and experiences outside the course), students could be graded on their ability to track progress and update their plans with more accurate estimates as the project unfolds. Otherwise, teams may learn to deny reality, lie about progress, and compromise quality to make their initial estimates appear accurate, instead of learning to dynamically manage their schedules and risks with real data.

4.1.5 Teach to the culture

Software engineering students typically view team skills as “flaky,” “soft,” “airy-fairy” subject matter, not to be taken seriously. Many do not view these skills as genuine software skills, despite their alleged importance for software development. Courses aiming to cover this material are often criticized for being either too vague or deliberately complicated to disguise the subject’s perceived illegitimacy.

By contrast, what engineers appreciate (confirmed by the personality preferences of UBC’s third year computer science students, discussed in Chapter 5) are courses that use no-nonsense concrete terms and examples, courses that respect the time-is-money philosophy and pack a lot of information into a compact straight-forward lesson that is kept clear and simple. If something is important, back it up with data and
hard evidence, graphs and numbers. Be specific. Provide objective measures for monitoring and assessing the skills. Relate the material to their immediate needs and concerns. Where possible, involve someone from within the culture to present the material or examples; learners are more likely to listen to and respect the opinions of someone who has already "gotten their hands dirty," than someone from another discipline who may not understand the realities of software development.

Taking this to another level, educators can successfully combat resistance by acknowledging that software engineers are already experts, only unaware of how to transfer their abilities and knowledge to team situations (i.e. surely those who know how to get two inanimate machines to communicate effectively should be able to learn how to do the same themselves!). Educators can take advantage of this existing expertise by using language and analogies drawn directly from the culture; for example, skills can be presented as "algorithms," "simulations" can be used instead of "role plays," concepts and notations of object-oriented design transfer incredibly well to the process of identifying roles, responsibilities and communication "interfaces" within the team, and asymptotic $O$-notation can be used to demonstrate the efficiency of different asynchronous decision-making "protocols." The idea is not to become ridiculous, but rather to work always from a position of strength, recognizing and applying the skills and knowledge software engineers already have to the development of team skills.

In this way, cultural change may result from team skills development, but is not a necessary prerequisite to improving competency. Software engineers do not have to enjoy "touchy-feely" issues to be highly effective team players. Fighting the culture or forcing cultural change will only impede progress.

4.2 Can people learn team skills from an ineffective team experience?

Is it safe to assume that both pleasant and painful team experiences provide satisfactory opportunities for developing team skills? Teaching would be simpler if, when setting up a team project, educators did not have to concern themselves with performance factors that could potentially interfere with a team's chances of succeeding.
Yes, it is possible to learn positive team skills, even when on a bad team, so long as learners have an opportunity to reflect on what went wrong and why, are enlightened as to what should have been done differently, and are given an incentive to plan for themselves how they could improve the next time.

However, the problem with bad team experiences is that they make it equally likely for participants to learn survival strategies that work against team performance, instead of positive team skills. For example, whereas one would hope students learn the importance of balancing workloads, students could just as easily become experts at avoiding work; if the team experience is very unpleasant, they may learn that it is better to skip meetings, to lie, to become hostile when asked to do something so that others eventually stop asking, to deny reality, or to routinely dissociate oneself from discussion during meetings. Alternatively, students might learn that it is always best to take on all the work themselves because they have not learned to trust and empower others to do a good enough job; this is a dangerous strategy because it does not build the team’s capacity to perform well when the keen member burns out or when a deadline draws near. Finally, if the team is so dysfunctional that it never gets beyond the forming or storming stage (of the team development life cycle), members will never get a chance to practice skills of norming, performing or adjourning.

Educators should therefore make an effort to set up teams for success, so that they have every possible chance to be effective. Particular attention can be paid to the context factors, cohesiveness factors, composition factors, and challenge factors of Chapter 2. Students can learn team skills even when their team is ineffective, but taking the extra effort to make teams effective will reduce the likelihood that students learn dysfunctional survival strategies in the process. Additionally, if the learning environment supports, reinforces, prompts and facilitates competent group work on a task, then students will be able to gain an appreciation of the benefits and qualities of superior teams and will have more opportunity to learn and practice skills relevant only to superior software teams.
4.3 Other skills for educators and trainers

So far, we have identified the following skills and strategies for people teaching team skills:

- Design the learning experience to accommodate practice opportunities, monitoring, formal training and assessment. For team skills, this should include multiple team project experiences.
- Support the teams' contexts and other performance factors so that every team has the potential of becoming highly effective.
- Promote mastery.
- Expect imperfection and do not penalize students for an initially low skill level.
- Teach to the culture.

Additionally, educators and trainers may have to work on their own team skills. "It may be difficult to develop skills in others which we lack ourselves, and many transferable skills may be outside the areas where we feel competent or confident. We may need to develop some of our own skills as part of the process of helping students to develop their skills." [Gibbs et al., 1994, p. 9]. Doing so will enable those teaching team skills to model appropriate behaviours, and to recognize and identify the competencies of students.
Case study: Software teams in CPSC 319

CPSC 319 is an undergraduate software engineering course at the University of British Columbia. The course's purpose ever since it was first taught in 1990 has been to give students a chance to work as a team to develop a complex software application. Its eight-month project is supposed to teach them to cope with not only the technical issues of employing mature software engineering techniques on a large project, but also the social coordination issues of working in a medium-sized team.

Unfortunately, when I became involved with the course as a teaching assistant in 1996 I discovered that students were learning other lessons:

"Team projects are the worst!"

"Software engineering is useless."

"I could have accomplished more by myself than we did as a group."

"Meetings are a waste of time."

These are some of the attitudes expressed by students and other teaching assistants upon completing CPSC 319 prior to the course’s 1996-97 offering. Are these the lessons we want them to learn? Statements such as these indicate that software engineering students are not learning in school to appreciate the advantages of teamwork over working alone, nor are they experiencing first-hand what it is like to work on an effective software team. This is of great concern because most students taking this course are just one year away from being junior members of industrial software teams, and aside from...
CPSC 319 there is no other course that all UBC computer science students must take prior to graduating in which they are guaranteed to experience working in a group.

Because I was convinced that team skills are extremely important for software engineers, I proposed some experimental changes to CPSC 319. This chapter describes my initiatives and general observations about the role team skills played in the student experience. As well, the personality preferences of the class are surveyed to better understand the composition of software teams and how members’ personalities might influence team processes. The chapter then closes with a few recommendations for improving CPSC 319. (Readers unfamiliar with CPSC 319 or the undergraduate demographics at UBC are encouraged to read a more detailed overview of the course in the Appendix.)

5.1 Overview of how team skills were previously taught

The UBC calendar is explicit about the course’s team project component, describing CPSC 319 as:

| 319(3) Software Engineering Project – The design, implementation, and test of a large software system, using a team approach. |

In keeping with the calendar description, some team skills were taught in past offerings of the course. The group project provided students with an opportunity to participate in meetings and work in a team outside of class. Additionally, CPSC 319 formally introduced several software-specific team skills during the lectures. These typically included skills of using style guides, make-files and source code control, and debugging or testing other people’s code. Additionally, students were required to practice giving and receiving technical feedback by exchanging and critiquing each other’s user interface designs, which they were then expected to modify according to their peers’ recommendations.

Nevertheless, despite this software-specific team skills training, the overall team experience was negative, and many students left the course without sufficient knowledge or skills for building their own effective software teams, unaware of having learned any general-purpose team skills. Part of the problem appeared...
to be that students were thrown into groups of eight to ten members and expected to learn and master most team skills by trial-and-error. However, as the course project served as a typical UBC computer science student’s first (and only) undergraduate team experience, many did not know what to strive for, did not have even basic skills to build on, and did not know what an effective software team was supposed to be like nor how it was supposed to operate. In a sense, they did not have enough team skills to develop team skills; to get started they needed extra help beyond what we were providing.

With insufficient skills for building an effective team, some students grew to hate each other by the end of the project. Team interaction was not reputed to be particularly fun or enjoyable. Most meetings were inefficient and unproductive. Workloads were poorly balanced within each team; the level of commitment varied a lot from member to member and a minority of members tended to complete the majority of the work. Few actions were taken in the past to specifically increase group cohesiveness (other than to guarantee that all team members shared one time slot per week when they could meet); in fact, during some course offerings, actions were taken that directly and significantly reduced cohesiveness (members were switched around partway through the project with the intent of balancing the teams and forcing some teamwork and process issues). Bad feelings about the team experience transferred to group work and software engineering in general, and interfered with the learning of team skills. The students’ dissatisfaction was reflected in consistently scathing course, instructor and teaching assistant evaluations.

5.2 Team skills taught in 1996-97

While CPSC 319 students may have learned that software engineering projects are the very antithesis of positive team experiences, it does not have to be that way. Shaw and Tomayko [1991, p. 51] describe a very different software engineering project course experience at Carnegie Mellon University:

“The feeling of accomplishment, the experience of learning to work together as a team, the use of skills previously exercised only in limited situations, all contribute to an exceptionally positive learning experience. Our former students often tell us that the only college friends
they keep in touch with are their team-mates from the software engineering course. Their positive feelings are reflected in significantly higher teaching ratings for the instructor."

This was entirely contrary to what we were experiencing at UBC; however, it indicated that 319 could be improved. In my role as teaching assistant, I devised and implemented some changes by teaching several classes, by interacting with the other teaching assistants and instructor, and by observing and working with individual students and teams over the eight-month duration of the course. The focus of team skills training was shifted to the more general-purpose team skills identified in Chapter 3, and attention was paid to all four areas of the skill learning cycle by introducing not only extra opportunities to use the skills but also formal training, monitoring and assessment. Table 11 summarizes the initiatives as they correspond to specific team skills identified in this thesis.

Additionally, several elements of the teams' “contexts” were altered to support teamwork. The course implemented its own web server so that teams could have sufficient common disk space for their project with shared access restricted to team members and course staff. The responsibility of specifying when each intermediate project deliverable would be due and how much each would contribute to the final project grade was passed on to the teams, who were also given more choices in which lifecycle model, development methodology, documentation and system features they would develop. Teams were provided access to private meeting rooms in one of the two computer science buildings on campus, for use during their scheduled meeting times, equipped with whiteboards, tables and movable chairs. Each team also received a “team toolkit” at the beginning of the year consisting of basic meeting supplies (paper, pens, transparencies, tape, etc.).

The main things that did not change for logistical reasons were the way teams were assigned, and the software application being developed: students studied a text-based user interface (UI) to a bug-ridden airline reservation system, and then each team rebuilt it from scratch, adding enhancements like a graphical UI, a database back-end, and more useful functionality. The interface between the UI and the back-end was provided to force some integration issues, and make teams' components interchangeable.

"The Importance of Team Skills for Software Development – by Carolyn Wick"
<table>
<thead>
<tr>
<th>Team skill</th>
<th>Formal (in class) Training</th>
<th>Practice Opportunities</th>
<th>Monitoring &amp; Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Project management</td>
<td>Overviewed the skills with examples of artifacts from a Project Management Plan (PMP).</td>
<td>Teams developed a PMP as part of their proposal, and were told to track progress.</td>
<td>PMP’s were marked &amp; teams were evaluated based on ability to follow original PMP.</td>
</tr>
<tr>
<td>2. Making meetings effective</td>
<td>Modelled an informational meeting in class and showed a video of ineffective meetings.</td>
<td>Weekly meetings.</td>
<td>Provided checklists for teams to monitor own effectiveness; T.A.’s graded the Meeting Basics.</td>
</tr>
<tr>
<td>3. Interaction</td>
<td>n/a</td>
<td>Increased team autonomy &amp; freedom of choice in setting goals and choosing methods for the project, which forced teams to make more decisions, solve more problems and manage more conflicts in meetings.</td>
<td>n/a</td>
</tr>
<tr>
<td>4. Group problem solving and decision making</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Conflict management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Understanding the user or customer</td>
<td>Handed out a paper on the importance of knowing the user and developing usage scenarios; overviewed the basics of the airline and travel industries.</td>
<td>Visited Travel Cuts, and played up the “real-world” project scenario (e.g. by using money as a metaphor for marks).</td>
<td>Usability was evaluated, but by the T.A.’s, not real travel agents.</td>
</tr>
<tr>
<td>7. Asynchronous group communication</td>
<td>n/a</td>
<td>Teams given access to e-mail, Vgroups (a private newsgroup-type forum) &amp; a course newsgroup.</td>
<td>n/a</td>
</tr>
<tr>
<td>8. Team building</td>
<td>n/a</td>
<td>T.A.’s led team-building exercises in first few meetings &amp; encouraged teams to develop team identity (team web page, name, logo).</td>
<td>Quiz question asked for team members’ names; Questionnaire asked students to reflect on and qualify the value of the team-building activities.</td>
</tr>
<tr>
<td>9. Process awareness</td>
<td>Taught the principles of CMM-based process improvement; explored how personality affects process &amp; team dynamics.</td>
<td>Proposal, PMP, and Questionnaires helped students reflect on and define their processes.</td>
<td>Personal logs contributed to each student’s individual part of team mark.</td>
</tr>
<tr>
<td>10. Other skills:</td>
<td>Taught tips for making effective presentations.</td>
<td>Teams had to present their proposals.</td>
<td>Presentations were assigned a team grade.</td>
</tr>
<tr>
<td>Oral presentations</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11. Initiatives for teaching software team skills in CPSC 319 (1996-97)
5.3 Observations on the teaching of team skills

How did these initiatives influence student attitudes toward the course and the development of team skills? Teaching team skills definitely set the tone for the course and set it apart from other project courses students had experienced. Initiatives to teach meeting skills, team building, and how personality can affect process all left a profound and positive impression on students. Some went so far as to credit the course’s success entirely to the teaching of these skills:

“I have worked in a team, but it was just “here are your partners – now get to it.” Nothing formal like agendas, minutes, weekly meetings, etc. I feel much better working with others now.”

“...I’ve heard that in the past, 319 was a lot different. That groups just programmed and it was just a course to make a large program and there weren’t any meetings, agendas, minutes, or teamwork for that matter.... I’m glad I took it this year and it has become the course it is now. I’ve heard rumors that 319 is a “pointless” course and if all you did was programmed straight for 8 months and earned 3.0 credits, I can see why it was called that... Perhaps the rumors will change when we all leave 319 this year... @”

Interestingly enough, the course always involved a team project with weekly meetings, and teams always had the option of developing their own agendas, minutes and teamwork. However, the general-purpose team skills that can be learned in CPSC 319 had never previously been explicitly defined; this obviously was an important contributing factor in our successful teaching of team skills in 1996-97, and leads to the following adaptation of familiar advice for all trainers and educators: tell them what they should expect to learn, teach them, and then tell them what you taught them.

Some other lessons learned from the experience are as follows.
5.3.1 Evidence of the need for team skills

The overwhelming need for certain skills above others was reinforced by students. The lack of group interaction, problem-solving, decision-making and conflict-management skills created a lot of problems in team communication, workload balancing, consensus and member satisfaction. One example of this includes the following:

"I always speak to fill in the empty silence. Our team has trouble with consensus as most people don't seem to be able to follow the issues. Our team has seething conflicts beneath the surface. Things are not getting resolved, only post-poned. We need to resolve things in our team. Things float too much without decision. Perhaps we need a leader. Our T.A. is not giving us feedback, he has missed most of our meetings and been away. We could really use feedback. ... I want more feedback on our waffling meetings."

Another student had the following insight into why many members of a team do not contribute equally in meetings and how this has resulted in poor collaboration and low quality decisions:

"There are team members who are over-bearing and cannot see that they are scaring (discouraging) other members from speaking out. They do not understand that some people have to be treated differently so that they will feel comfortable talking. Personally, I find it hard to speak because others speak too "powerfully." They do not understand how discouraging their words are. One member often tries to become a facilitator even though that member is not the official facilitator. .... It [the proposal] is not finished and I do not know what I am doing. Again, the overbearing ones are controlling the group. They don't see their mistakes. I think we are on the WRONG track. ...the overbearing members tend to take control and suppress the shy members. The decisions were based on basically one person. UNFAIR. When I am facilitator, I will set an example of how one should manage a team and show the errors in the overbearing members' styles. Overall, I am not VERY COMFY WITH THE PROPOSAL."

Many more desperate cries for help can be found in the comments in the Appendix, many of which come from students who would have benefited had we addressed interaction, consensus decision-making and conflict management skills in class.
Additionally, development of project management skills should be given higher priority in the curriculum. Enormous amounts of anxiety were induced by students not knowing how to control the project schedule, track progress or manage risks. The problem we discovered was that project management skills cannot adequately be taught in an hour lecture, and yet are extremely important for teams to have autonomy in completing their work.

5.3.2 Students learned what we taught

This may seem obvious in retrospect, but the skills students mastered were the ones we taught, practised, monitored and assessed. Skills that were not supported through all four stages of the experiential learning cycle did not develop, although the activities introduced to support some parts of the cycle may have benefited the teams in other ways.

For example, the Meeting Basics were mastered by all teams within just one or two weeks (i.e. two or three iterations of the learning cycle) – a perfect example of how effective it can be to follow this learning model. (This actually became a problem in itself for T.A.’s when all teams started achieving 100%, bringing to light the conflict between encouraging mastery and achieving a normal grade distribution.) However, despite this success, team meetings continued to be ineffective. The problem was not in the way the skills were taught, but in the way they were defined. Recall that meeting skills were defined to help make meetings more efficient; for meetings to be truly effective, students would need group interaction, problem-solving, decision-making and conflict management skills as well. Those other skills, as evidenced by Table 11, were not supported to the same extent as meeting skills.

Team-building skills, by contrast, were not mastered by all students, even though every group was led through a series of team-building activities and assigned team-building tasks early in the project. The team building (which included icebreakers to get acquainted, exercises to develop a common identity through creating a team name, logo and web page, and activities to become more cohesive by having fun together exploring teamwork issues) did serve the intended function of helping teams build cohesion and
move more quickly through the forming stage. However, because we failed to teach students why team-building was important, some students missed the point, as evidenced by responses to a question of whether or not they benefited from the team-building activities done in the first few meetings:

- "Not really, but it's fun."
- "No, not really. But it was a nice way to 'break the ice.'"
- "I am not able to see any direct benefit to the straw castle exercise."
- "No. I thought those activities were nonsense."

(It is worth noting however that the majority of students, 110 out of 123 respondents, did claim to have benefited from the team-building activities. Their comments are included in the Appendix.) The point here is that experiencing use of a skill is alone not enough to develop the skill; in this case, doing team building with each team was like feeding them a fish for a day, whereas teaching them how and why to do it themselves would have been more akin to teaching them to fish for life. Perhaps this is why many companies who have brought external facilitators in to do team building for a day or weekend retreat have not experienced long-term benefits from the activities.

5.3.3 Prior team experience cannot be assumed

Feedback confirmed that many students had not worked in a team before taking CPSC 319:

- Uncomfortable, nervous at first, since I'm not used to this. Much ok now.
- No, I've never worked in a team situation before. I think it will be great to get all kinds of ideas from other people.
- This is a first time that I am working in a team. It is very good in a sense that it is very close to real world, and I think it'll prepare me for the real job market.
- I think this course should be a good experience of real world situations. But I hope you consider that it is our first experience and mark us considering that please.

A general lack of experience was further evidenced in the class's response to a video used to teach meeting skills. The entertaining video "Meetings, Bloody Meetings," shown at the beginning of the year
to demonstrate how not to run a meeting, was very poorly received. This surprised me since many authors recommend it [Laiken, 1994, p. 68; Gibbs, 1995b, p. 42], and the instructors of a similar project course at Camosun College reported in an interview that students there thoroughly enjoyed it and learned from it. I realized that to appreciate the video’s humour, one needs to have attended similarly ineffective business meetings already. Unlike many of the college students who had already worked in industry, CPSC 319 students possibly did not yet have the experience at the start to appreciate the lessons of the film nor to find the film funny. Furthermore, the film did not show them what they most needed to learn: how efficient, productive and energizing an effective meeting could be. Therefore, while I still believe this particular video and its sequel can be valuable teaching aids when shown to people with some meeting experience, instructors of CPSC 319 should assume that many students entering their class will not have this prior experience unless team skills are introduced earlier in the UBC computer science curriculum.

5.3.4 Teams were too big

Teams of eight and up were too large. Many split into sub-groups, some into opposing factions. Most did not become particularly cohesive. The main tasks could not easily be divided up among so many people without introducing extra tasks to give everyone an identifiable visible product to work on. In teams that did not add extra work for themselves in this way, individual contributions were very hard to evaluate and assess.

Yet, the most significant problem was that one hour once a week was simply not enough to give every team member a chance to speak during a meeting when issues needed to be resolved. The larger teams did not have the option of building consensus; instead, they had to adopt more expedient but less effective strategies, such as voting, to “resolve” conflict. Consider the following student comments:

- *I have worked in a team before, usually 5 or 6 members. I found 10 in a team is a lot of people to coordinate during a 1 hour meeting and to coordinate the work sharing outside of class.*
• One hour a week is too short, we’ve been meeting twice a week so everyone has a chance to think about the problems and has time to speak.

• Not enough time for meetings. One hour per week for 8 people is realistically not enough.

• [Consensus?] Not always; the time is never enough

• Sometimes we have conflicts, but we always resolve the conflict by vote anyway.

• We usually have an agreement. If consensus can not be reached, we vote....

• I think some members in our team are unrealistic in the amount of work they think we can finish. ... [Any major conflicts?] Yes... some of us wanted to do just the GUI. But majority wanted to do both. Resolved? I guess since our proposal says we’re doing both, it’s been resolved.

• Finally, while developing skills of working in large teams is arguably important for software engineers, it is inappropriately difficult for a student’s first team experience.

I recommend teams no larger than five or six in the future, so that meetings require coordinating at most seven people’s contributions when the teaching assistant is present.

5.3.5 The Hawthorne Effect

One reason many new initiatives within 319 were successful was because we ran the course as an experiment, as a pilot project, and the entire class knew they were guinea pigs. Knowing their instructors cared enough about the course to experiment, solicit and receive feedback, make ongoing changes and respond to developments was highly motivational for students. Several comments illustrate this:

• “I think this questionnaire was helpful and that we should HAVE MORE of them. A greater class participation would make classes more interesting. Overall, I LIKE THIS COURSE.”

• “I think it is very important that a course get feedback like this from the students, especially when the course is a project. (Please voice this to the Dept.) The best thing about this course is that it’s open enough for the students to learn as much as s/he can/wants to. No limits. Lots of research. The fallback is obviously those who don’t do any work. I think the drastic change of the way 319 has been is very effective. It’s a great success.”
While this has likely interfered with our ability to determine whether the introduction of new technology and skills really improved team performance or whether the improvement was merely an example of the Hawthorne effect (where the simple act of conducting an experiment produces the effect under study), in some ways it doesn’t matter. Even if our experimental results are skewed, they are skewed on the high side, which is what we are after anyway. Anything that improves team performance, student motivation and learning is likely to help students develop team skills and positive attitudes toward software teams. I simply recommend that anyone who wants to take this research a step further and introduce similar concepts in their classroom do so with an open mind and flexible agenda, making students partners in deciding what content will be covered and how.

5.4 New understandings of software team performance

In addition to teaching us more about teaching team skills, the 319 experience enabled us simply to observe a large number of software teams in a contrived setting and verify the importance of the many performance factors and skills identified in the earlier chapters of this thesis.

5.4.1 On the effect of giving teams more freedom

As we saw in Chapter 2, teams need autonomy in completing their work in order to become highly effective. Giving teams the power and authority to set their own goals, choose their own methods and implement their own decisions can significantly influence team performance. Additionally, it can place a greater demand on teams to develop skills of technical leadership, project management, integrative problem solving, group decision making, conflict management, and more effective interaction.

In our approach, instead of giving teams freedom to choose their own projects while providing structure and guidance in the processes to be used by all teams, we did the opposite: we let them choose their own process for developing a specified product. Unfortunately, not only did this prove to be more challenging, but we did not teach students the necessary skills for coping with the challenge. Now that we understand
the experiential learning cycle, it is easy to understand in retrospect why this approach to the students' first team project led to incredible frustration:

- "It is very frustrating! It is not realistic at all to let a group of inexperienced students write a proposal like this. We have no idea how much docs should be produced (how large are they)? How many lines of code are we going to write? We are going to use tools we never learned, or even heard of! It might be more practical to introduce us how, in the real world, the S/W company does this. Go through every stage together with T.A. (supposed to be experienced). Then do the project next term by our own."

Nevertheless, most students became acutely aware of software process, did develop some project management skills, and indicated no desire to reduce the amount of freedom despite the challenges:

- Too many [good things] to list. We need more freedom in this course. I don’t fall for the argument about it being too hard to mark projects if we have too much freedom in what we will do. If that were true no one would ever write essays in history or English classes.

Additionally, many teams seemed to become more cohesive as a result of having to establish a common vision for their project that satisfied every member’s needs.

5.4.2 How does involving representative users affect the performance of software teams?

Initially, I wanted to see if introducing students to real, representative, potential users of their system would help in developing team skills and in producing a positive overall team experience. In principle, such initiatives should make a difference. Involving users can give teams a stronger sense of purpose, which should motivate students, enhance their feelings of accomplishment, and provide more incentive for building quality and usability into the end-product. Additionally, when the user community includes people other than themselves, students may develop skills of communicating with people in other application domains, practice identifying concrete, detailed examples of tasks users perform that the system should support, and gain an appreciation that the user community can think and act very differently from developers, and even differently from how developers expect they will think and act.
Therefore, I set out to find some real, representative, potential users of the students’ airline reservation system (ARS). I arranged a site visit to Travel Cuts, a travel agency on the UBC campus, for a group of 12 volunteer students. During the visit, students watched travel agents go through both real and imaginary scenarios, and at each step the agents stopped to explain what they were doing and what problems they sometimes encountered. The agents described their vision for a new system. Students had opportunities to ask questions and several brought cameras hoping to share impressions of the travel agents’ workplace with team-mates back in the classroom. Students unable to participate in the visit were encouraged (but not required) to visit other agencies on their own and submit a report as one of their project deliverables, for which their team would receive credit (one team did this).

Student feedback was overwhelmingly positive. Participants enjoyed the experience, and all went out of their way to show their appreciation to the travel agents and me for arranging it. Participants commented that it made the course and project more realistic and relevant. Students also acquired substantial domain knowledge. For example, they were surprised to learn that travel agents often liked using the command line and cryptic commands (which were fast, powerful and learnt in school).

However, the experience did have some unexpected drawbacks. Inconsistencies between what the instructors had asked students to develop and what a real airline or travel agent would likely have requested were emphasized by the visit. In order to make sense of the conflicting information, students had to keep the two sets of requirements separate, adding an extra layer of complexity to an already complex project, and limiting the value of the real travel agents’ input.

For example, to make the site visit relevant, we suggested that the real agents’ comments about the commercial system could represent what imaginary users would say about the students’ ARS, even though the two systems had few specific features in common. When the real agents pointed out features

---

1 Participation was voluntary, but we recommended that each team designing a graphical UI send a representative to a travel agency. The group of 12 who visited Travel Cuts represented less than 10% of the class, but close to 50% of the teams, and supposedly they relayed what they learned to their team-mates and possibly to other teams.
of the real system that they liked and wanted to keep in an upgrade, students could assume either that: (1) imaginary users wanted to see those same specific features in the students’ ARS (which meant changing the system substantially), or (2) imaginary users felt the same way toward the students’ ARS as the real agents felt about the real system (which meant avoiding any major changes). Thus, many teams assigned a high priority to making their new graphical UI’s match the course’s previous character-based UI as closely as possible because the agents had stated that such a consistency would be important to them. However, in grading the projects, teaching assistants expressed disappointment that so few teams adopted innovative new ways of designing their UI’s and effectively penalized teams that did not make the functions more “usable” (from a non-travel agent’s perspective!).

The worst problem encountered though was that the course’s half-way support of user-centred design contributed to one team’s demise. Two members of this team initially become so highly motivated by the possibility of developing a real product for a real customer, that they convinced their team to adopt a design and development environment different from the rest of the class so that they could easily extend their system to meet the real travel agents’ needs after the course ended, as a possible summer job. However, while the plan could have saved them work in the long run, it made the course project significantly more difficult. The team encountered numerous difficulties with their implementation and received little help from the course staff in supporting their chosen development environment. The more experienced members ended up dominating the task activity. The excessive workload probably caused their other courses to suffer, and because of the way the project was evaluated, their marks in 319 suffered too since they could not satisfy all of our requirements. A year later, they still had not implemented a variation of their system for Travel Cuts, probably because the team had no remaining desire or capability to work together again after the conflict-ridden 319 experience.

Thus, involving representative users significantly influenced team members’ motivation and learning; however, it added another layer of complexity to an already complex project. As well, for most teams, decisions were ultimately based on the expectations of the person who would be evaluating the product (in our case, the instructors), rather than on the needs of the person who would be using the system down
the road (real and imaginary travel agents included). I expect that whenever customer and user expectations are in conflict, most software teams would react in a similar way.

5.4.3 What is the predominant software development personality?

Perhaps the lack of team skills in industry and our failure to adequately teach them to software engineering students in school is due in part to the unique software development personality: do we have the necessary personality preferences for working in teams and for carrying out positive team behaviours?

The stereotype

The reported shortage of team skills certainly may not surprise people familiar with the stereotypes (i.e. how could introverted male computer geeks without social skills work effectively on a team?). According to Fisher et al. [1997, p. 109] the stereotype is "clearly the myopic, narrowly focused, young male who sits at his computer all day." They summarize student impressions (aptly termed "Geek Mythology") about computer science at Carnegie Mellon University:

- "computer science students have a single-minded focus and talk incessantly about computing"
- "CS is the department with the really smart students"
- "the work load is extremely heavy (with special emphasis on the amount of time that it takes to complete programming assignments)"
- "most CS students...[are] immature males who burrow into their computers for all forms of satisfaction"

Is there any basis for the stereotype? If so, how might such personality characteristics affect teamwork and the development of team skills among software engineers?

The survey

I surveyed the personality preferences of students in CPSC 319 during the 1996-97 course offering, and found that many students did not match the stereotype. Nevertheless, the survey did suggest that a CPSC
319 student’s preferences are likely to be consistent with the stereotype along several dimensions. Furthermore, a CPSC 319 student is considerably more likely than someone randomly selected from the general Western population to have preferences for introversion rather than extraversion, giving added support to the stereotype. Above all, the data indicates that our future software engineers have diverse personality preferences that may impact how teams operate and how team skills should be taught.

The survey that produced these findings involved having the students complete a short in-class questionnaire based on Carl Jung’s theory of personality type (a copy of the questionnaire is included in the Appendix). This questionnaire is a simple variant of the Myers-Briggs Type Indicator (MBTI), which is extensively used in career counselling and development, business and education for helping people explore their personalities in a number of areas. The four orthogonal dimensions along which Jung and the questionnaire differentiate personality are defined as follows by Noring [1993]:

1. **Energizing Scale:** How are you energized? Extraversion (E) is the preference for drawing energy from the outside world of people, activities or things. Introversion (I) is the preference for drawing energy from one’s internal world of ideas, emotions or impressions.

2. **Attending Scale:** What do you pay attention to? Sensing (S) is the preference for concentrating on information taken in through the five senses, and for paying attention to facts, details, material things, and what is real and actual today. Intuition (N) is the preference for taking information in through a “sixth sense” and for noticing why things are the way they are, how things might be, how things relate, what an experience means or implies, and the possibilities of tomorrow.

3. **Deciding Scale:** How do you prefer to make decisions? Thinking (T) is the preference for organizing and structuring information in a logical, objective way to make fair decisions according to a set of rules. Feeling (F) is the preference for organizing and structuring information in a personal, value-oriented, subjective way to make merciful decisions that reflect the importance of choices for oneself and other people.
4. **Closure Scale:** What is your preferred life style? Judgement (J) is the preference for living a planned and organized life, where things are brought to closure, and where one aims to bring everything under control. Perception (P) is the preference for living a spontaneous and flexible life, where things are left open-ended, and where one’s aim is to understand and adapt to events rather than control them.

In each of the four scales, most people are capable of behaving either way depending on the circumstances, but every person usually has a preference for one of the two opposite choices. The preference is designated by a letter (‘X’ is used when the preference is unclear or unknown).

**The results**

The personality preferences of the entire class present on the day the questionnaire was distributed were collected (n=93 students). These totals can be compared against the approximate percentages of the total population (in Western culture) who hold each preference (population data is taken from Noring [1993]).

A summary of the results is shown in Figure 8.

<table>
<thead>
<tr>
<th></th>
<th>Sensation-Thinkers</th>
<th>Sensation-Feelers</th>
<th>Intuitive-Feelers</th>
<th>Intuitive-Thinkers</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTP</td>
<td>1 (1.1%)</td>
<td>(n/a)</td>
<td>XNTX</td>
<td>1 (1.1%)</td>
</tr>
<tr>
<td>ISTJ</td>
<td>10 (10.8%)</td>
<td>6 (6%)</td>
<td>ISFJ</td>
<td>7 (7.5%)</td>
</tr>
<tr>
<td>ISTP</td>
<td>10 (10.8%)</td>
<td>6 (6%)</td>
<td>INFJ</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>ESTP</td>
<td>4 (4.3%)</td>
<td>12 (13%)</td>
<td>ESFP</td>
<td>5 (5.4%)</td>
</tr>
<tr>
<td>ESTJ</td>
<td>11 (11.8%)</td>
<td>12 (13%)</td>
<td>ENFP</td>
<td>4 (4.3%)</td>
</tr>
<tr>
<td>Sensation-Thinkers</td>
<td>36 (38.7%)</td>
<td>35 (38%)</td>
<td>Sensation-Feelers</td>
<td>23 (24.7%)</td>
</tr>
<tr>
<td>Sensation-Feelers</td>
<td>23 (24.7%)</td>
<td>35 (38%)</td>
<td>Intuitive-Feelers</td>
<td>13 (14.0%)</td>
</tr>
<tr>
<td>Introverts</td>
<td>32 (34.8%)</td>
<td>25 (27.2%)</td>
<td>Extroverts</td>
<td>37 (39.8%)</td>
</tr>
</tbody>
</table>

**Key:**

- = Number of students in the class who recorded this preference (out of 93)

- = Expected number of students based on % of population with this preference

**Figure 8.** Personality preferences recorded by students in CPSC 319 (February 25, 1997)
The implications

Because CPSC 319 is offered once per year and is mandatory for computer science students, and because most computer science students find jobs in the software industry upon graduation, my survey sample would seem to be a reasonable representation of the types of people who will be entering the British Columbia workforce in the next few years. Some characteristics of the student data were compared against the population (using chi-squared analysis). What the data reveals about software teams follows.

- Software engineers are more likely to be Introverts than Extraverts.

Furthermore, the number of “I” types recorded in class was considerably more than expected based on the population data (p<0.0005), as shown in Figure 9.

This suggests that communication is likely to be a problem in software teams. With over twice as many “I” types represented here compared with the general population, it is likely that the greater proportion of people in software teams may feel uncomfortable contributing to discussion, may be reluctant to express authentic support for good ideas or confront others in constructive disagreement, and may prefer to think a lot before they act, sometimes without wanting to act at all. Furthermore, if we assume that “E” types will form a minority in most software teams, as they did in our survey, then many software development
groups are likely to see the few “E” types dominate meetings, be more aggressive, or take charge, especially without many other extraverted types to challenge them. Software teams may benefit from periodically having people write down suggestions to be collected and read out, or from splitting up into smaller groups before having a spokesperson present back to the team to facilitate participation of shy members. Otherwise, if a member ends up feeling dominated by other members, feels they have no opportunity to participate in discussions, or if their suggestions are often ignored or overridden, they may lose interest, a sense of hopelessness can result, and they may become apathetic to the team’s goals. In fact, this is exactly what we observed in 319.

• “N” types are less common among software engineers than “S” types, but more common in this group than in the general population. “T” types are more common than “F” types and also more common than expected.

The two scales (S-N and T-F) together define the software engineer’s unique problem-solving style [Hellriegel et al., 1995, pp. 114-130]. The S-N scale determines what a person pays attention to, with implications for how they will gather information. The T-F scale determines how a person will evaluate the gathered information to make a decision. The combination results in four problem-solving styles (ST, NT, SF, NF). Software engineers in CPSC 319 are twice as likely to be “NT’s” (p<0.005) and a third less likely to be “SF’s” (p<0.025) than in a randomly chosen group of the same size from the Western population (illustrated in Figure 10).

These problem-solving styles can affect an individual’s behaviour in teams [Hellriegel et al., 1995, p. 127]. “ST” and “NT” types, being thinkers, may want to make decisions objectively, and may get frustrated when there is no clear logical best choice, or when there is a definite logical best choice that some “F” types in the team do not acknowledge. “NT”s may be perfectionists and in a team may pressure others to put off decisions rather than risk making a mistake; however, they also tend to be particularly good at discovery, inquiry and problem-solving which is consistent with desirable traits for software engineers. Meanwhile, “SF’s” are the ones who tend to be best at detecting and resolving interpersonal
problems within the team; the lack of this type in 319 compared with what was expected may explain why conflicts were so prevalent.

![Diagram](image)

**Figure 10. Problem-solving styles in CPSC 319 vs. in the Western Population**

Additionally, the data may help explain why process awareness and process improvement are not skills naturally found in software teams. "S" types may help keep teams on task, pinned down to reality, and may be good at focussing on the details of a deliverable or solution; however, "S" types may be less likely to take a step back from their work and reassess the direction, consider alternatives, or question why they are doing what they are doing. These latter points are typical strengths of "N" types, the minority.

### 5.5 Recommendations for CPSC 319

We did not succeed at pleasing everybody; several students hated 319 during the 1996-97 offering, just as their many predecessors did before them. However, we do have the benefit of considerable feedback from students, and since we now understand the factors influencing team performance, we are in a better position to comment on what likely caused many of the problems and what could be improved. Some recommendations for issues that still need to be resolved are as follows.
Recommendation #1:

Make sure the team's task and team size are defined such that the work can be easily partitioned into easily identifiable and interdependent subtasks that give every team member something meaningful to do.

Rationale:

One of the reasons for assigning up to ten members per team was to increase the total amount of effort available for the project so that students could participate in developing something significant. However, the tasks themselves did not require so many people, and were very hard to divide up so that each team member could make an identifiable contribution. Students perceived no clear reason to work together as a team other than to distribute the workload among more individuals, which required more effort to coordinate. Many groups had only one or two people working on the project at any one time. In effect, eight people were expected to do in eight months what one or two people could do in eight months (or possibly even in one month). Students did not experience interdependence, did not learn to value the diverse skills and ideas each member brings to the team, nor saw evidence of positive synergy: eight people in eight months producing a better product than one person alone could in sixty-four months.

Recommendation #2:

Team skills should be introduced earlier in the computer science curriculum.

Rationale:

To do well in the project, students need to know a lot of material before they begin. Ideally they would already have covered the basics of software engineering in CPSC 310: development life-cycle models, software process improvement, specification techniques, methods of high and low level software design, how to specify the system architecture, how to do testing and quality assurance, issues of maintenance and designing for portability and reuse, and how to do documentation. However, even if the co-requisite course is able to adequately cover all of these software engineering principles and techniques before
students start the project, 319 requires other concepts and skills as well. Students need applied technical programming skills that go beyond languages, data structures and algorithms (as this is their first large group project, they often do not yet know how to use make-files, source code control, and other programming techniques and tools needed on a multi-person programming project). For the Airline Reservation System project, they need domain knowledge about the airline industry and its applications, database management skills, some client-server concepts, and user interface design knowledge. Coordinating eight people over eight months requires project management skills, such as workload balancing (estimation, scheduling, role and responsibility allocation), risk management, alternatives for organizing their development team, and basic progress tracking. Finally, students need group communication and interaction skills (possibly for the first time), including how to run and participate in meetings, make effective presentations, give and receive feedback, select and use group problem-solving and decision-making strategies, and manage conflict. There is not enough time to adequately introduce and cover all of these topics in 319 before students begin the project, especially since the administrative details of the course and project have historically taken up most of the lecture time; yet, these skills are an essential component of any team project’s success.

**Recommendation #3:**

*Additional team experiences should follow CPSC 319 in the curriculum.*

**Rationale:**

To master team skills, students need to participate in multiple team experiences, obtain regular feedback on how they are doing, and have reason and opportunity to plan ways to improve. CPSC 319 cannot realistically give all 150 students in the class all the personal feedback they need. Past course instructors deserve an enormous amount of credit for successfully managing such a large number of undergraduates, teams, and development resources. Nevertheless, the large class size makes it impossible for the instructor to interact with all students and teams to give them desperately needed individual advice and guidance. Graduate student teaching assistants help alleviate much of the workload by sitting in on team
meetings, coaching student teams, and grading deliverables. However, we cannot expect all computer science teaching assistants to arrive with sufficient software engineering, project management, facilitation, and feedback skills to do their jobs effectively and consistently because they are after all products of the same system that has been found faulty in these areas. Furthermore, even if there were resources for training the teaching assistants, there is no time to do so before the course begins because many of them are new graduate students who arrive to begin their own studies only in the same week as they must begin their teaching assistant duties. If students participated in multiple team project experiences distributed throughout their degrees, they would have more opportunity to develop skills and there would be far less pressure on 319 to do everything.

**Recommendation #4:**

*Increase the workload intensity of the project, possibly by coordinating CPSC 319 with other courses.*

**Rationale:**

Year after year, students suggest that the course be worth six credits instead of three, which leads one to believe that the course workload may consistently be too high. However, given the observed ineffectiveness of the course’s project teams, an alternate cause for the problem may be that the workload is not intense enough. While there are no strict rules for instructors, the Registrar’s office suggests that students plan for two hours of lab or study time outside of class per credit hour per week for a one-semester course, and so it is reasonable to expect a regular three-credit course at UBC to occupy 104 hours of a student’s time, including classes, plus whatever time they choose to spend during the exam period. After subtracting up to 26 weeks of lectures and team meetings, students in CPSC 319 would each be left with only 52 hours to work on the project, study for quizzes, and complete any other course assignments. For the project, this is equivalent to less than one-and-a-half weeks of full-time work spread out over eight months! Teams that limit themselves to the required workload must spread their effort so thinly (and often unevenly) that they find it hard to achieve and maintain momentum, do not spend enough time together or on the project to build a highly cohesive group, waste time reminding themselves...
of decisions made long ago and since forgotten, lose sight of their long-term project goals, and wait a long
time to see results and obtain feedback. Additionally, because the teams are so large, the allotted time is
often insufficient for each member to communicate ideas in meetings, or for teams to resolve conflicts by
consensus. Effective teams that avoid these problems often go way beyond the required workload (many
reported putting over 200 hours of work per person into the course in 1996-97), which explains why the
end-of-year evaluations contain so many requests for making the course worth more credits. Any attempt
to coordinate CPSC 319 with other courses in the curriculum (such as by focusing entirely on team skills
in 319 and letting the individual members’ contributions to the database, networking, or user interface
components of a one or two-month team project count toward their grades in those other courses) could
help in balancing student workloads better across courses so that students can devote a more intense effort
to the team project, without it detracting from their other courses.

Recommendation #5:

CPSC 319 should itself continue efforts to teach general-purpose team skills.

Rationale:

Whether or not this was the best way to go about teaching team skills, the initiatives served to improve
the course in the eyes of many students who provided feedback about the class and the team experience. I
will end this chapter with selected positive comments about the course in general:

- I enjoy working with others and I think it is very useful to have this experience because out in the
real world you rarely work alone, usually a team is involved. I think being able to work with
others is a necessary skill, just like math, or programming in C++, or English, it is a useful skill
to have whether you like it or not.

- It makes the class more exciting – more than all the other CS classes – because there is more
interaction, and the situation is more “realistic.”

- I have always felt that it is an important part of career preparation; and now I am actually
participating in this activity, I believe it is really worthwhile.
• The lectures I appreciated the most were the ones about general professional skills: how to run a meeting (good!), Gantt and PERT charts, how to give presentations. I would love to have more of these next term (time permitting) since I think they are of the most lasting value.

• I think it is a really good chance of learning team work. I think this should go on.

• I really prefer to work alone. Group work, though, is a good useful experience for me (to get out of my comfort zone).

• It’s great!! There should be more courses like this, with teamwork and interactivity.

• Good. Computer Science is desperately in need of providing its students with human interaction.

• I think the faculty should spend more time preparing students in a team oriented environment.

• I like this course.

• 319 is one of the best courses in CPSC.

• Great course, glad I waited!

• Thanks!

The Importance of Team Skills for Software Development – by Carolyn Wick
Chapter 6

Conclusions

Software teams that can satisfy their customers' expectations without compromising their own team members' needs or the future capabilities of the team are becoming increasingly important in the high-technology industry. This thesis has outlined many of the skills necessary for making teams more effective, including skills for team leaders, skills for team members, and skills for teaching the skills to team leaders and members.

First of all, we showed that software team leaders will benefit from understanding all the factors that influence team performance. Knowing which software engineering best practices and tools have contributed to other team successes and failures is definitely an asset. However, this knowledge alone is not enough to make a team effective. Leaders need to be able to help the team get started, manage conflict, become cohesive, and bring tasks to closure. Leaders need to understand how the team composition can influence performance, and what factors to consider when selecting people to join the group. Leaders need to know how to design an appropriate challenge for the team, whose corresponding goals and tasks are sufficiently understood, motivating, realistic, attainable and interdependent. Leaders need to be able to set up a supportive context, and if necessary, change the physical work setting, tools, evaluation and reward structures, and the team's autonomy to facilitate high performance. And leaders need to suggest effective operational and teamwork processes for the team to follow that facilitate process improvement. These and other performance factors were described in detail in Chapter 2.
Secondly, we saw that software team members can benefit from numerous team skills as well. In addition to bringing many important software-specific skills to the task, members can help their teams become more effective when they have general-purpose skills in the areas of project management, making meetings efficient and productive, communicating and interacting effectively (both in person and asynchronously), managing conflict, solving problems and making decisions as a group, choosing appropriate team-building activities, understanding the user or customer, being aware of process, and leadership. These skills were detailed in Chapter 3.

Thirdly, we identified teaching team skills as one way to mitigate the perceived shortage of these skills in the software industry. Skills for teaching team skills effectively were proposed in Chapter 4.

Finally, research was conducted within an undergraduate software engineering course at the University of British Columbia to observe the importance of all of the above team skills for student software teams, as well as to try out some new ways of integrating team skills training into the curriculum. The case study, described in Chapter 5, confirmed that software teams are heavily populated with a certain type of person whose personality preferences can affect team processes. As well, the experience reinforced the need to teach, monitor and assess skills of conflict management, effective interaction and leadership, group decision making, integrative problem solving, and project management.

To determine where to go next with this research requires understanding how this thesis relates to other computer science research done in the past. The discipline of developing effective software teams (software team engineering) is inherently interdisciplinary, pulling together the fields of software engineering, organizational behaviour, project management, process improvement, and human-computer interaction (see Figure 11). Each of these other areas has something to contribute to the field:

- **Software engineering** is the technological and managerial discipline of producing and maintaining software products within time and cost constraints. It contributes knowledge of the task, tools and development methodologies facing the software team.
• The area of human-computer interaction contributes knowledge and practices for satisfying users, customers and other stakeholders. Techniques and principles of user-centred software development and participatory design can make teams more effective by helping them communicate with clients to develop products that satisfy the expectations and needs of those who will receive or review them.

Figure 11. Future research in software team engineering will be inherently interdisciplinary

• Organizational behaviour contributes an understanding of human behaviour, attitudes and performance in organizations. It is itself interdisciplinary, drawing on concepts from social and clinical psychology, sociology, cultural anthropology, industrial engineering, and organizational psychology [Hellriegel et al., 1995].

• Project management contributes practical guidelines for managing team processes in the areas of planning (identifying tasks, sizing software, estimating effort, scheduling tasks, budgeting),
organizing (staffing, allocating roles and responsibilities, selecting team structures and processes, managing communications), directing (work authorization, replanning, motivating and evaluating personnel), managing risk, and monitoring (using reviews, quality assurance, progress metrics, rate charts, and earned value measurements).

- Process improvement, which focuses on reducing variability across development experiences over time, has many similarities with efforts to understand and control variability in effectiveness across teams, and contributes principles and techniques for organizational learning and continuous improvement.

Software team engineering is based on the premise that teams “do not grow haphazardly, nor is maturity automatic” [Bradford, 1976, p. 29]; instead, teams must be designed and developed over time with hard work and disciplined processes benefiting from acquired high-level skill sets and tools, much like how software itself is engineered. Because the skills required for building effective software teams also need to be developed systematically over time, software team engineering is a necessary component in a software engineering curriculum.

Future research in this area should strive to develop more process guidelines for team leaders, members and educators, so that they can achieve greater control over team effectiveness, explore how these practices interact with common software development methodologies, and develop tools to support leaders and members in their team improvement activities. In these ways, we may eventually gain control over productivity, quality, time taken, errors made, satisfaction, and other measurable outcomes that vary substantially across software teams today.
Bibliography


*The Importance of Team Skills for Software Development – by Carolyn Wick*


Hughes Aircraft Canada (1993). Hughes ATC Overview Course: Excerpts from ATC overview course given by Transport Canada.


The Importance of Team Skills for Software Development – by Carolyn Wick


Video Arts (1993). *Meetings, Bloody Meetings, and More Bloody Meetings*. (Instructional videos of common mistakes in running business meetings, starring John Cleese, approx. 30 minutes each. Available at UBC's Sedgewick Library, call number HF 5549.5 C6 M437.)


Appendix I: Structure of UBC's CPSC 319 (1996-97)

The UBC calendar describes the course as:

| 319(3) | Software Engineering Project – The design, implementation, and test of a large software system, using a team approach. |

Attempts at integrating team skills training into CPSC 319 in 1996/97 were constrained by several course characteristics. At that time, the class demographics, co-requisite course, length of course and workload, staffing levels, and process of assigning students to teams were as follows.

**Demographics**

The class is very large: we typically have around 150 students enrolled, as was the case in 1996/97. The course is mandatory for Computer Science majors. Most students take it in their third year of a four-year undergraduate computer science degree, although each year a handful of students are in their fourth year or higher, or come from other departments such as Math, Physics, Commerce, or Engineering. With one course offering per year, it is a fairly representative cross-section of the undergraduate computer science program, which in 1996/97 was approximately 25% female. During the 1996/97 offering, 85% of the students in the course fell in the eighteen-to-twenty-five year age group, signifying they were in their first post-secondary degree; however, the range of ages represented in the course extended into the 41-50 year range [Page, 1997, p. 41]. Some students already had work experience in the software industry, such as those enrolled in UBC’s cooperative education program. Most students are Canadian but a significant number come from Asia and have difficulty with the English language.
Co-requisite course

Another software engineering course, CPSC 310, is meant to teach isolated techniques, methodologies, and principles of software engineering; 319's intent is to give students an opportunity to apply and integrate these concepts and technical skills on a large team project. By expecting students to take CPSC 310 before or during the first semester of the project course, 319 was supposed to be able to focus entirely on the project. However, the two courses have only been loosely coordinated in the past, and some time has typically had to be reserved in 319 anyway for teaching various concepts and skills required by the project.

Length of course and workload

The course duration is eight months (September to April), but earns students only three credits. Thus the total workload per student is supposed to be no more than for a regular course, but stretched out over twice as much time, partly to guarantee adequate time for covering prerequisite material in CPSC 310 before the project begins. The class meets during one scheduled lecture hour per week for the two consecutive semesters (each consisting of thirteen weeks of class). Students also register for one of five hour-long tutorial time slots used for weekly project team meetings which a teaching assistant may attend. 319 does not have a final exam.

Staffing

The course is administered by one instructor and three to five graduate student teaching assistants who are hired to help with up to twelve hours of marking, teaching, and student-interaction per week for the course's eight-month duration. In teaching resources, this is actually double what is typically allocated for a three-credit course.
The process for assigning students to teams

Students did not have the option of selecting their own team-mates. Teams were formed the same way as in previous years: students registered for one of five tutorial time slots when they signed up for the course, and then during the first week filled out on-line questionnaires self-assessing their skills in various technical areas. An automated routine allocated them to a team based on their preferred tutorial (team meeting) time slot, while distributing skills as evenly as possible across teams. Team sizes ranged from six to ten members, with an average of eight. In 1996/97 we had eighteen teams each paired with one of four teaching assistants for the duration of the project. Teams were assigned by the third week of class, although the composition of some teams changed again when students dropped the course later in the term. (For a copy of the questionnaire and more detail on the groups' compositions, see Page [1997, Chapter 2 and Appendix A].)

Grading Scheme

Student grades were based on a combination of individual and group marks. A typical grading scheme for each student prior to 1996-97 was as follows:

• 1/3 of final grade = individual effort on the project (necessarily a more subjective measure made by the teaching assistant).
• 1/3 of final grade = individual achievement (intended to be a more objective measure based on student’s contribution to the group project).
• 1/3 of final grade = group achievement (also intended to be a more objective measure based on the entire team’s final product).

Because of the substantial project, the course had no final exam. It has also been recommended on several occasions that the course be pass/fail, though this has not been adopted to date.
Appendix II: Selected student feedback

The following comment was posted by a student to the course newsgroup two-thirds of the way through the course:

"...I've heard that in the past, 319 was a lot different. That groups just programmed and it was just a course to make a large program and there weren't any meetings, agendas, minutes, or teamwork for that matter. Though unconfirmed (since when you're in first year, you look up to third year courses and hear just rumors about them), I'm glad I took it this year and it has become the course it is now. I've heard rumors that 319 is a "pointless" course and if all you did was programmed straight for 8 months and earned 3.0 credits, I can see why it was called that... Perhaps the rumors will change when we all leave 319 this year... :-)"

Halfway through the year we gave the class a questionnaire. 123 students filled in the 8-page questionnaire during class and submitted their feedback. Some responses to some of the questions are as follows. (Responses have been grouped by similar themes.)

| Many of you would have never worked in a team situation before. |
| How do you now feel about working with others? |

**COMMENTS ON THE VALUE OF TEAM PROJECTS AND SKILLS FOR CAREER DEVELOPMENT:**

- I like the idea of "trying" to work in a team situation. It will surely help me in real life situations after graduating.
- I enjoy working with others and I think it is very useful to have this experience because out in the real world you rarely work alone, usually a team is involved. I think being able to work with others is a necessary skill, just like math, or programming in C++, or English, it is useful skill to have whether you like it or not.
- I have always felt that it is an important part of career preparation; and now I am actually participating in this activity, I believe it is really worthwhile.
- Working with others is a good learning experience, for future jobs. It's a very good idea since software is often produced in a team environment.
- Working with others can learn a lot of things that we cannot get from lectures.
- I feel that working with others is important because in reality one should know of what to expect from group members. Projects in the working world are pretty much group oriented and to know what to expect from a group will help out in future group projects/meetings.
- This is a first time that I am working in a team. It is very good in a sense that it is very close to real world, and I think it'll prepare me for the real job market.

*The Importance of Team Skills for Software Development – by Carolyn Wick*
• Working in a team is a great experience, and will generally benefit us when we go out and look for a job.
• I think it is a really good chance of learning team work. I think this should go on.
• It’s great!! There should be more courses like this, with teamwork and interactivity.
• I think the faculty should spend more time preparing students in a team oriented environment.
• Good. Computer Science is desperately in need of providing its students with human interaction.
• That’s good. And it is the best way to gain experience of the real working format in the Computer Software Industry I think.

COMPARISONS TO PREVIOUS TEAM EXPERIENCES:
• No, I’ve never worked in a team situation before. I think it will be great to get all kinds of ideas from other people.
• I have worked in a team, but it was just “here are your partners – now get to it.” Nothing formal like agendas, minutes, weekly meetings, etc. I feel much better working with others now.
• Although I’ve worked in a team situation before (planning club activities and youth group activities) I still consider the experience of CPSC 319 a valuable one.
• Just fine, but then, it’s not a new thing for me. Nice to see this sort of thing in CS though.
• I enjoy working with others – and after my co-op job, where I did work in a team atmosphere, I did look forward to working in a team again.
• I have worked before in a team environment (during my co-op work). So I’m quite comfortable and relaxed working with others.
• I have worked in teams before – always enjoyed it.
• I have done it many times before. I really enjoy working with people towards a final goal.

PROBLEMS OF ACCOUNTABILITY, DEPENDENCE ON OTHERS, WORKLOAD BALANCING:
• Good! It’s difficult though adjusting to the fact that you have to depend on others. Usually I like to do things on my own so I know it gets done to my satisfaction.
• Working in teams is very important in the real world. The main problem in a school setting is that there is a chance team members may not be willing to try hard. In the real world, these people would likely not be working with the company long! I don’t know how to rectify this, however.
• It’s nice to be able to delegate: reduced workload => reduced stress. However, there is the concern that teammates won’t always get their assigned work done on time/up to quality. In a work situation, where the group is a subgroup of the company, they would answer for that. Here, we do.
• Still don’t like it. Work is (unavoidably) partitioned unequally, and those people who try hard to produce quality products have to mesh it with the work of people who don’t care/didn’t try/aren’t skilled as much, which results in poor marks and a “Why bother?” attitude.
• Working in a team is good. It helps each one of us in dealing with other people once we are out in the real world working. However, sometimes I found out that one or two people are not working as much as the other people. This really disappoints me.
• What shall we do if one person doesn’t complete his job and doesn’t notify other members until the due time? And somebody doesn’t come to meetings quite a few times and every time he says he is sick. Is there anything other members can do?
• It can be fun. So far it is ok. Some people always end up doing more than others.
• Learning experience – hard to work together when everybody’s not on the same wavelength – not everyone has the same priorities for the course or will put in the same effort.
• It gave me an opportunity to do more than I could do on my own. Still sort of concerned about relying on other people, but it seems like things are going well.

The Importance of Team Skills for Software Development – by Carolyn Wick 113
• I have worked in teams before. It seems like some team members in my team are there (at meetings, etc.) because they have to be! They show very little enthusiasm and put very little effort into things. But I don’t think that has anything to do with this course and it seems like that is the way they are. I think that there should be a way to distinguish the “reward” of people who do more work, or somehow kick those people out and tell them to go and learn something that they like (little harsh 😞)

• It’s alright, although you have to rely on others (not always good)

ISSUES OF TEAM SIZE:

• The teams should be smaller (if possible). I know the point of this course is to get experience working in large teams, but this is most people’s first time. Teams of 6 would be a lot better.

• I have worked in a team before, usually 5 or 6 members. I found 10 in a team is a lot of people to coordinate during a 1 hour meeting and to coordinate the work sharing outside of class.

• In my group, I have to work with people who don’t care about the course and people who work very hard. This makes me worry because it concerns my fellow members and myself. Some of (few of) us are willing or motivated to go out and look for info on their own while others wait and expect those who have researched to tell them exactly what to do. Working with people and coordinating 8 people is a very hard task.

• We unfortunately (or fortunately) have the largest team (10 members) and this means that more is required of us, yet this provides the less “enthusiastic” members of our group an excuse to take more of a passive role in team affairs. So same number of people end up working (as in the smaller groups) but more is expected from our group.

• I like the change (from working individually to working in a group) but sometimes it gets complicated when there are so many people.

• Never worked in a team with members > 5 before. This is a new experience.

• A relatively small team seems to be quite easy to manage. I’d hate to be working with 8 or more people.

• I think that for this project if we had half the people we could get the job done faster.

ON THE TEAMS’ LACK OF AUTHORITY OVER THEIR OWN COMPOSITION:

• I would prefer to choose my own team members, or have the option of “peer evaluation” because we have some members who aren’t doing JACK $#!@#!!

• People should all be allowed to choose their members of people to work with.

• Didn’t like the large group size (8) and not being able to choose own group.

• This perhaps is more difficult [than] in real world since there is no authority among group members. You can’t hire, fire, resign.

SCHEDULING PROBLEMS:

• Working with other people is the most difficult task, coordinating the time schedule is the hardest thing. Working with others is a necessity in the real world, but to make things easier a very strong management is required.

• I like working with others, but it is hard with 5 other courses to allocate necessary time to meet, discuss, plan, etc. I do a lot of my work after midnight; unfortunately my group wasn’t too keen about having I am meetings.

COMMUNICATION:

• It is very difficult to keep on task and also to communicate effectively (e.g. people often have different interpretations of what is being discussed).

• Communication is really important.

• I like the team environment, yet I can’t help but feel frustrated with the team because we seem so inefficient. Because there are a lot of us and it seems like there’s always at least 1 person who “doesn’t understand” so we spend a lot of time going over and over the same topics.
GENDER ISSUES:
- Very interesting, really test my adaptation and attitude to others. Was a bit intimidated when I realized I am the only female in the group. Nonetheless, good teamwork doesn’t rely on gender balance but individual (personal) involvement, participation and enthusiasm. I like working with others.

LEADERSHIP:
- I think our team works well and everyone gets to make their points. There does however, appear to be a natural hierarchy forming (or already formed) that sees certain people “in charge” in the group.
- I think teams work best with a leader. Our team has no elected or appointed leader and so our trials in democratic rule tend to bog things down.
- It’s kinda fun. I think that some people are born leaders. I wish I had more leadership, I always seem more comfortable concentrating my energy on tasks rather than organization.
- It’s a good experience to be a team facilitator.
- In this case, it would be nicer to have roles assigned because no one really wants to be a team leader because of extra work and responsibility. In real world, leader with a higher salary is assigned role.
- Inconsistent facilitator leads to poor management. We really need a more consistent way to organize/facilitate the group. Perhaps have 4 out of 8 do it, but one per month?

CONFLICTS & GROUP DECISION-MAKING:
- I’m learning a lot about group dynamics and dealing with making group decisions.
- Can be frustrating when they hold you back or won’t recognize your ideas. Can be humbling when they point out why your ideas are wrong.
- Good and bad. Good side: I learned something from other people. Sharpened my communication skill. Bad side: Hard to come up with an answer than makes everybody happy. Sometimes, I have to do something I don’t really want to but that has been decided by the team.
- Good. Team conflicts and planning are the most important aspects of teamwork that we’ve encountered.
- Fine, but there are many conflicts between members.
- Fine. But sometimes we have some different opinions.
- Fun. Difficult to make decisions. More new ideas come up.

ON SKILL DEVELOPMENT:
- It has given me the opportunity to test my so-called “team player” skills. It’s made me realize that when you are working on some large project, there may be things that could go out of control.
- I think it’s good in working as a team because people can share work and improve cooperation. Also, each team member knows who’s good at what and who’s weak in what so that the whole team can balance.
- Good experience, good for practicing skills i.e. time management, social skills, learning from others.
- I realized that I am a team player and I do enjoy working with others.

EXPRESSIONS OF DISCOMFORT IN WORKING WITH OTHERS:
- A lot of fun, but sometimes it makes me a little bit embarrassed.
- Working in a team requires responsibility and gives me too much pressure.
- Uncomfortable, nervous at first, since I’m not used to this. Much ok now.
- Fairly comfortable – though I should probably speak more.
- Not very comfortable.
- Before, I was not quite comfortable in working in a team, but in this course, I discover the advantages in working in a team.
• I really prefer to work alone. Group work, though, is a good useful experience for me (to get out of my comfort zone).

• Not very comfortable. I prefer to work alone.

• Get a chance to know more friends. Kind of uncomfortable before but it is ok now 'cause we know each other better now.

• Not very used to it.

• I feel uncomfortable at first. But now I'm used to it.

• Good. Feeling more comfortable now.

• Feel much more comfortable about working with others [now].

• I feel a little pressure but working with others helps to finish a job in less time.

• It's practical experience. Excited. Sometimes worried because you don't know how well you'll work with others.

CHALLENGING EXPERIENCES:

• I think it's more challenging than what I imagined.

• I found that it's pretty challenging.

• Difficult.

• It was interesting. It was tough at first, but it did get better as we got to know each other.

• Working well in a team requires much practice and experience.

• I like it but it takes a lot of work/time/effort to make it work.

• It's little hard to work with people we don't know. But as time goes it is becoming easier.

• Not too bad. Things tend not to get done as expected, but in general, we got a lot done. ... It's kinda fun.

• As of now, working in a team situation has not posed any significant problems. However problems will probably occur later on.

POSITIVE EXPERIENCES:

• It's fun, different from the normal routine, lets you develop team skills.

• Working with others is great as long as everyone is friendly and cooperative. I'm fortunate that my team is composed of such people.

• I find it very interesting to observe the team dynamic. I'm learning a lot.

• Interesting and exciting to work with others. Learned a lot from my group members.

• I like it. My teammates are great.

• Personally, I like working with other people. I find people interesting. I think that there are a whole set of new difficulties to face with a team. Some of these things are pinning us down, but I am learning from the experience.

• There are much more things to consider than when working alone. But it's [a] pleasure to work with somebody responsible.

• Working with people who are motivated and fulfill their commitments is a great deal of fun.

• I feel very good about working with my team members. Every team member is cooperative.

• It's fun because I get to exchange my ideas with others and at the same time learn a lot from others.

• Feel comfortable. Everybody participates in a very informal manner, like discussion around the dinner table.

• I really like it. Nice team members and working together is always less stressful and more enjoyable.
• Quite interesting. People are nice.
• Working with others is a lot of fun.
• Good experience.
• Working with team members just like the daily working. I think it is good. But I also should have a lot of patience, participation, encouraging to other people. Until now, I feel good.
• Allows people to learn more from others close to them. Good.
• Manageable.
• It seems to be ok.
• It’s ok for me.
• It’s cool.
• It’s fun.
• It is fun and gain a lot of experience.
• Comfortable – depends on team personalities more than anything else.
• Fine!

Do you have consensus about things in your team, and do you feel that you are allowed adequate time to speak in your meetings? Has your team had any major conflicts, and have they been able to resolve them? Have your T.A.’s comments about your team work been helpful?

UNEQUAL PARTICIPATION IN MEETINGS:
• We agree fairly quickly on most items and I have enough time to speak, but I am concerned about others not saying anything to avoid work.
• I think I am allowed adequate time to speak in my meetings, but sometimes people don’t hear what I say!
• I think our team is ok. There are a couple members that still don’t speak up. I try to encourage them by directly asking questions. Some people are shy and need extra push. Me personally I sometimes feel I speak too much since when nobody has what to say I usually do. I think [the T.A.] should be more specific and comment more. Maybe the T.A. should privately encourage the quiet members to speak. I think that more comments are needed (good & bad).
• I believe that it is a problem when those who speak English as a 2nd language do not speak their minds during meetings, but rather remain silent throughout. I have no suggestions on how to remedy this, but you... should be aware of this.
• Some choose not to speak. We can’t force them to if they don’t want to.
• We do have consensus; however, the majority of the talking is sometimes done by the more vocal members of the group. The quieter people in the group wanted more say, and were given an opportunity to do so.
• I feel like I am always forcing people to talk. Most people in my team seem very bored and do not talk much. No conflicts, except people do not volunteer to do things (agenda, meeting, etc.) Everyone has lots of work (all CompSci people have lots of work).
• I always speak to fill in the empty silence. Our team has trouble with consensus as most people don’t seem to be able to follow the issues. Our team has seething conflicts beneath the surface. Things are not getting resolved, only post-poned. We need to resolve things in our team. Things float too much without decision. Perhaps we need a leader. Our T.A. is not giving us feedback, he has missed most of our meetings and been away. We could really use feedback. ... I want more feedback on our waffling meetings.
DOMINATING MEMBERS:

- One person tends to dominate the meetings. I do get adequate time to speak, though.
- Certain people do seem to dominate the meetings, but I get the impression that if they didn’t, we’d just be staring at the walls.
- Yes, I feel that we have consensus… but then I am also the one with the biggest mouth. I thus try very hard to keep it shut out of respect for the facilitator, and reserve my comments/input for detailed input.
- There are team members who are over-bearing and cannot see that they are scaring (discouraging) other members from speaking out. They do not understand that some people have to be treated differently so that they will feel comfortable talking. Personally, I find it hard to speak because others speak too “powerfully.” They do not understand how discouraging their words are. One member often tries to become a facilitator even though that member is not the official facilitator.
- It [the proposal] is not finished and I do not know what I am doing. Again, the overbearing ones are controlling the group. They don’t see their mistakes. I think we are on the WRONG track. …the overbearing members tend to take control and suppress the shy members. The decisions were based on basically one person. UNFAIR. When I am facilitator, I will set an example of how one should manage a team and show the errors in the overbearing members’ styles. Overall, I am not VERY COMFY WITH THE PROPOSAL.

SOURCES OF CONFLICT:

- Every member in the team is allowed to express their opinion freely. But that’s where the conflicts came from.
- We haven’t had many conflicts so far. I can foresee one though. There seems to be a lot of “tiptoeing” around issues (i.e. deciding who is going to do what). When nobody steps up and takes charge to say “I’ll do that!” some people on our team have a real problem with delegation. It’s kind of annoying.
- Consensus – not always. There is always one person (not the same person each time) who wants something done a different way. Always plenty of time to speak; especially since a few of the team members are quiet.
- One member is not pulling his weight in our team. We hope the situation will improve.
- Major conflicts are the dominating personalities colliding. Eventually, they managed to reason out their differences.
- So far I have been able to speak most of the time and also everybody in my team has been allowed to speak adequately. So far there has not been any conflict in my team. [but later notes:] My team has decided to do both RMS and GUI, but I believe we should have proposed to work on only one section.
- We’ve had meetings and decisions made, but not everyone agree[s] on the decisions (and they don’t say anything about this. They complain afterwards). What can we do about this?
- Team meeting is really important to me. But the skill of every member is not all at the same level, … hard to assign position[s].

PROBLEMS OF ASYNCHRONOUS COMMUNICATION:

- We did not have conflicts. However, I think the problem is somebody takes a long time to reply [to] the mail and sometimes nobody replies [to] the mail.

HOW CONFLICTS ARE RESOLVED:

- We have had conflicts over (1) which life cycle model to use and (2) who will be involved (or how many people) in the implementation. For (1), because we were presented with two models, people figured we were limited to these two. False. People were not prepared at the meetings – They did not understand the differences between the models, hence dragged the meeting on. Solution: one person went off voluntarily to look up details and presented this to the group. For (2), one person wants to do everything, while others want to do implementation. This ended in a half-hour argument and resulted in having those who want to implement really badly [do] the job, but not necessarily the best people for the job.
- No conflicts, but a hard time recognizing or addressing problems. Not knowing how to tackle them once we recognize them.
• We usually have an agreement. If consensus can not be reached, we vote. Unless one of the minority has major objections (not just a simple preference) then the majority decision is accepted. Any conflicts that cannot satisfactorily be settled with a vote are discussed. We decide what we need to do to make a decision and assign tasks so that the decision can be made at the next meeting. We have not had any conflicts that could not be resolved within 2 weeks.

• Sometimes, but we always resolve the conflict by vote anyway.

• I think some members in our team are unrealistic in the amount of work they think we can finish. ... [Any major conflicts?] Yes... some of us wanted to do just the GUI. But majority wanted to do both. Resolved? I guess since our proposal says we’re doing both, it’s been resolved.

• There was an issue raised whether we should do both parts of ARS and it’s now been solved, though I still don’t agree with the decision, but I’ll have to go with the team’s decision.

• Most of the time our team reaches a consensus but that does not always happen immediately. Often things are discussed and different points of view are presented and debated before consensus can be reached. No major conflicts so far.

• The freedom given has also given rise to restrictions. Restrictions in the 319 sense of group members. I believe that only counselling will help... nothing really can be done. Group problems should be solved within the group, but outside influence is welcomed in case the problem(s) is too large.

• I’m comfortable because all the concerns and opinions I had I voiced to the group.

ISSUES OF MANAGING MEETING TIME:

• One hour a week is too short, we’ve been meeting twice a week so everyone has a chance to think about the problems and has time to speak.

• Not enough time for meetings. One hour per week for 8 people is realistically not enough.

• Hard to coordinate activities. One hour meeting a week is not enough.

• [Consensus?] Not always; the time is never enough to resolve the problems that arise.

• The team meetings are too short. What we must do is to come prepared to meetings knowing what the agenda items are and what decisions we have to make. However, not everyone is willing to do that. In fact, we have had two 3-hour meetings outside of school already, and halfway through there is almost [a] couple of people who doze off and are simply waiting to leave. On top of that, not everyone participates, and the meeting ends up being one person dictating his/her ideas to the whole group, which results in intimidation.

• Usually there is consensus in our meetings after discussion of the issues. Our meetings often run long, and this is due mostly to a great amount of material to cover, but also due to the fact that we need to improve our meeting skills.

• Yes I feel I have enough time to speak my mind in meetings but this is done by not strictly following the allocated time as the agendas indicated. Not very good, but it’s irresistibly easy to side track. I think facilitator has to reinforce time constraint.

• The biggest conflict is 1 hr/week is not enough and we still have trouble agreeing on a place for 2 hr meetings.

• A longer meeting is needed.... [Also] it’s sometimes hard to force some members to attend team meetings and meeting on time.

• Maybe the meeting is a bit short.

ON HAVING THE T.A. PRESENT DURING TEAM MEETINGS:

• It is very informative to have a “third” party present to watch and comment on our meeting. Our T.A. has made several suggestions that have helped our group dynamics.

• I think T.A. should attend every team meeting.

• I feel more comfortable when we have the meeting without T.A.

The Importance of Team Skills for Software Development – by Carolyn Wick
COMMENTS RELATING TO THE ASSESSMENT AND FEEDBACK:

• Still worried about people not pulling their own weight (why work hard if the group gets the same mark anyway?)

• [T.A. comments helpful?] Somewhat. They are useful – only because the info is given to us afterwards. Many times we needed his input/guidance but he was not there. (Because he’s not required to be at every meeting). Comments on what we did wrong is definitely useful, but comments on how we can do things right would be better.

• Still don’t like teams. Despise the entire concept of “group mark.”

• Yes they [the T.A.’s comments] have [been helpful] but some of the expectations are hard to fulfill. I think that decision-making is easier in a more informal atmosphere: the time constraints during the meetings (i.e. times allotted for agenda items) and the pressure to conform to them are a little stifling.

• [Have our T.A.’s comments been helpful?] Not really. We get relatively low marks, along with comments that suggest that if we lie, we’ll get better marks: i.e. 6/10 … [because] agenda time estimates were wrong (for first three meetings).

• Having a 10-minute in-class quiz worth ½ as much as the 25-hour threads specification was teeth-gnashingly infuriating.

• Not having a final exam is better because our work is not just based on an exam.

• We haven’t received too many comments about our “team work.”

Any general comments about working in a team?

• It is like any job. There is work to be done and decisions have to be made. I enjoy it like work. Sometimes fun and sometimes a real pain.

• Our team turned out to work ok.

• Often fun and informative…many heads are better than one.

• No major conflicts… but this is being said on the basis that most of my team members are quite talkative and outspoken people. Everyone in the team has been told earlier in this team that ‘speak your mind’ is required to produce cohesive meeting.

• If you don’t say anything…not much gets done. You have to give your input.

• It will always be hard to get along, but in essence the more diverse a team is [with] regard to characteristics and idea-making, the more ideas will surface, therefore improving the group… if problems are solved.

• Working in a team means compromise and participation. If every team member tries actively to get involved, the team will be more productive.

• I think that it’s always hard in team situations because leaders are few and hard to come by. Also, many times people are embarrassed and shy in front of people before they become accustomed to each other, which usually results in little or no work accomplished in the early stages.

• I like working in a team. I met a lot of people that I would not have otherwise. I think the frustration I feel at times will make it easier to deal with similar circumstances later in life.

• We need more control in a meeting – right now, one person tends to take over (not the facilitator). But how? We need to figure out what’s wrong with our meetings, why people aren’t motivated, how we can motivate each other, build more trust among us. How? Need a workshop....

• Still don’t like it. When I mentioned (last year) how much everyone dreaded 319, I was told industry wants even more courses like it. So that’s who the university is run for? If they pay my tuition, then they have a say.

The Importance of Team Skills for Software Development – by Carolyn Wick
Lectures: Are there any lecture topics that you would like us to cover next term, or that we should have covered earlier?

- The lectures I appreciated the most were the ones about general professional skills: how to run a meeting (good!), Gantt and PERT charts, how to give presentations. I would love to have more of these next term (time permitting) since I think they are of the most lasting value. CM [configuration management] might be a good topic for a lecture next term: e.g. different types of CM, models (exclusive locking vs. branch-and-merge,...) and managing software releases (i.e. release engineering, whatever that is).

- Management skills: how to deal with irresponsible team members.
- How to deal with team conflicts.
- Ideas on how to deal with team conflicts and non-performing team members.
- Do a workshop about working in a team.
- Project management stuff.
- The project management is not very clear to me. We still need more lectures.
- Ways to share the work need to be learned and it would help if the T.A.'s/instructor could give us some concrete ways.
- Other real world software engineering project examples to motivate me!
- The lecture hour could be used more efficiently. Perhaps some video tapes showing how teams divide work load of a large project. Or the cycles that a project goes through would be a good choice.
- Management skills (coordinating everything). Emphasize differences among real life, this course, generic UBC course. (Possibly video?) insight on how real companies do things professionally. Non-co-op students don't seem to be visualizing the real world very well.
- More on the tools/skills we need. If possible, maybe can show us more real-life examples (case examples, videos,...); it's important to know that we are learning something that's actually being used in real-world.
- What would be neat is talking about a real company (I mean, existing); examples of project management and their successes. I guess this is edging towards commerce ☹.
- I only wish that SE process could be reviewed briefly, since a lot of people who took 310 last year forget about SE mostly already.
- More information regarding real life situations would make things more interesting. (e.g. real-life group projects and examples). Field trips? Hee Hee... have members from each group speak for 5 minutes each ☹.
- How about a video on what we're doing, but in a real life setting? That should give us a feeling of how, say, proposals should be done.
- I would like to know more things about operations in a real computer company. How does every worker work together?
- Some [lectures] aren't really that helpful: i.e. we spent a ton of time talking about meetings. It didn't really change our efficiency.
- The Meeting Basics lectures should be covered before the first meeting so we know what to expect/how to prepare.
- Some lectures are a bit silly, i.e. the one on meetings, the whole thing could have been done more briefly. But some are overloaded with information, i.e. the one on threads. Try and balance them out.
- I like the technical topics like Oracle, SUIT, in more detail. Maybe in more than one lecture.
- The "about SUIT, Oracle, et al" lecture could have been a little earlier, since there were times when we had to make decisions about which to use before we even knew what they really were.
• Tcl/Tk, Oracle, C++, SUIT, HTML, ...

• Making the notes available on the web is a good idea.

• I think the lectures are pretty useless because we can gain the exact info through the web.

• Lectures are sometimes boring, although the materials taught in class might be useful.

• I think the lecture topics have been great so far.

---

**The proposal: Having almost finished the proposal, do you know what work you will be doing next year?**

**Do you feel comfortable with what your team has decided to do next term? ... Any general comments about the proposal?**

---

• It is very frustrating! It is not realistic at all to let a group of inexperienced students write a proposal like this. We have no idea how much docs should be produced (how large are they)? How many lines of code are we going to write? We are going to use tools we never learned, or even heard of! It might be more practical to introduce us how, in the real world, the S/W company does this. Go through every stage together with T.A. (supposed to be experienced). Then do the project next term by our own.

• The proposal, and the help, is too wishy-washy. We have no experience as a group, estimating time, using different software packages, etc. It seems to be a lot of guessing.

• Our proposal as it stands is too vague and general. We know too little about anything. Especially we don’t have enough knowledge with various tools we’re required to use, so it’s almost impossible to come up with an accurate and detailed plan or products. (Maybe courses in database and CPSC 310 should be strict prerequisite courses for this course).

• It is worth a large amount of our mark. I don’t understand how the 15% proposal, 25% for rest of project ratio was determined? The proposal is also a bit of a chicken and egg (chicken → project, egg → proposal ©). We have very little knowledge of what a chicken is or how to make one yet we have to create an egg. The only consolation is that everyone is in the same boat.

• I think my team tends to be a little too optimistic. It would be better to have some more guidance.

• I think we are too optimistic. Maybe overestimating our ability.

• No [I don’t feel comfortable with what our team has decided to do], I would like to be a main programmer, but it seems my “skills” are needed elsewhere, and I still think we are biting more off than we can chew.

• I’m a bit worried we’re in over our heads, but isn’t that part of the thrill of school. ... It’s just hard to anticipate and specifically plan everything. That is the challenge though.

• I think my team members are not being realistic about the size of the project, nor about its work load. From my experience in prior projects and course, I know for sure that completion of project will take much longer than the time assigned (100h/team member). But unfortunately, they don’t realize it. I feel that we’ll end up not finishing most of the functionalities.

• Have too much freedom in deciding the workload of project.

• It is very hard for us to decide what we are going to do next year since we still don’t know how busy our next term courses will be. It is hard to make a realistic timeline for the proposal.

• Yes [I feel comfortable with what our team has decided to do] since I don’t really know anything. I mean, I don’t know how to write Oracle, ndbm, etc., so I don’t know how hard the project is going to be.

• To me, I got the feeling that my team have underestimate the workload.

• I still don’t really have a firm grasp on what is supposed to be included in the proposal, whether it is a proposal to CSAir, or to CS319 markers, or both.

---

*The Importance of Team Skills for Software Development - by Carolyn Wick*
• We … have (still) no idea what we are getting into.
• It is helpful, but I HATED doing it (probably the thing I hated most this school year...actually, to be honest, my entire university career) [except maybe the Scheme self evaluator in CS 126]  
• It is very extremely difficult to decide on a life cycle model, methodology, tools to use, with hardly or no experience (except for 310) using them.
• I think we’ll be able to finish on schedule, but it’s hard to say since I really have no idea how long it takes to code a GUI and database. Since we don’t really have any prior experience in coding GUIs or databases, it was hard to estimate how long we need to allocate to each section. Information like, “last year each group took an average of 100 hours to code the GUI” would be helpful to give us an idea.

In this course, we have given you lots of freedom (compared with other courses), but with certain restrictions necessary to guide you through the work and to maintain consistency across teams. What are the good things about having freedom? What are the bad things?

• We get practice making decisions.
• Feels more like the real world.
• Builds planning skills.
• Creativity flourishes.
• Creative/wacky ideas can really liven up this boring project.
• Teaches responsibility…. We are allowed to decide for ourselves the direction our project will take.
• Each student will think about the topics a lot more than they would if everything was specified beforehand.
• Valuable project management experience.
• Too many [good things] to list. We need more freedom in this course. I don’t fall for the argument about it being too hard to mark projects if we have too much freedom in what we will do. If that were true no one would ever write essays in history or English classes.
• The only draw back to having CS people get together and plan a project seems to be that CS people (in general) usually like to leave things to the last minute. This can be a little problematic when certain work depends on the timely completion of other work. So I think it is good to have some deadlines and such.
• Hard to force procrastinators to do work (myself included).
• This freedom may cause some people to get a false sense of security in that they might think that they have less work than they do, and so they may have serious difficulties at deadlines.
• With poor time management on the team’s part, you can get really swamped with work.
• Can cause confusion and indecisiveness especially when paired with lack of experience.
• I would have liked to see a more “structured” approach to the lectures and course in general. I realize that this course is supposed to give a lot of freedom on what we do, but many times I found myself wondering “What do I do now???”
• How do we know if we’re doing enough work? Are we being judged against other teams? But we don’t know what other teams are doing? (i.e. [How are you planning to] “maintain consistency across teams”?)
• Very important: we do what we want and [can] not complain about…wanting to do something later (not forced upon us).
• We think that the project we are doing is more our original work. [But] the grading would be more difficult in the sense that there wouldn’t be so much consistency among teams.

The Importance of Team Skills for Software Development – by Carolyn Wick
We are allowed enough room to work the way we like and produce the work in our desired manner.

We get to decide what we want to do and when we want to do it. For a person such as I who likes to be told what to do and then goes and does it, this empowerment is a bit unsettling. However, I feel that one must learn to think for oneself and learn how to plan one’s own projects, so this empowerment is good.

I like the idea about using any language for implementing.

I like that we get to decide when/what/where/how things are done. But sometimes finding guidance for the things we are doing is difficult since the course is so open.

We should have been able to pick what project we wanted to implement. Maybe it could have been a list of 5-6 different projects, but we should have had a choice.

The freedom given has forced us to be creative as a group to develop our proposal where if more restrictive structure was followed I don’t think we would have needed to make so many group decisions.

Marking is still done on a narrow criteria and for all the freedom we still have to meet it. (I’ll admit it wasn’t as narrow as I thought it would be.)

Freedom on how the T.A.’s mark the work may cause some serious problems. Some teams may score high and the others may score low though they produce the same standard of work because the T.A.’s have different standards and measures to give marks away.

Too much freedom, lose sense of direction too easily, don’t have a clue what is happening sometimes. Never know if headed in right direction with work until it is almost too late. Course needs to be more structured so certain questions can be answered more in advance.

Ambiguity sometimes promotes ‘going on the wrong track’ on assignments.

Sometimes we are kind of lost in our work.

---

Did you benefit from the team building activities (e.g. building straw castles) that were done in the first few meetings?

- It was indeed a great idea to have that practice in the very beginning of the semester; during that straw-building activity (although only 15 minutes or so), we actually encountered most problems that we may actually face when working as a team.

- Yes, but we should have had 2 meetings the first 1-2 weeks for ice-breaking like this. Even in Nov., there are times when I feel the ice-breaking didn’t last long enough.

- Our team has yet to really break the ice. We needed more of an initial “Hurrah!” from the T.A. Most members are extremely shy.

- No, because it’s not long enough.

- They were only done in the second meeting for us. Yes, they were great. More should take place. The people I referred to [in a previous comment as being unenthusiastic and unmotivated to work in the team] actually showed some signs of enthusiasm there ☺.

- I did like the parts where we were introduced to the group by partners and picked a team name together.

- I think the team building activities helped us to get to know one another a bit better, and to practice interacting on a “small” project. It was good to have the accomplishment of seeing our plan succeed.

- Yes I got a rough idea of how to cooperate with others and produce a product.

- Yes it’s really wonderful that all of us have a common goal and I felt excited when we...achieved the goal.

- I did because we know how to divide up the labour force as well as cooperate with each other.

- I learned that co-operation is very important.

The Importance of Team Skills for Software Development – by Carolyn Wick
• I learned to compromise and try to see things in other people's perspective.
• I thought it was interesting how immediately people's personalities were revealed, so yes, I thought it was a good icebreaking activity.
• Yes I really got to see the personalities of the others when faced with a task.
• Yeah sort of. The leaders in the group showed up when we were building the castles.
• Yes I can see who has which potential skills in team situations, such as leadership, making decisions, and tolerance.
• Yes the straw castle was a great exercise that taught everyone something about the other members' personality and our group dynamics.
• Yes it's a nice way to know who are good team leaders and who can give good ideas.
• Yup. Found out who was vocal and who was not. Allowed some of us to figure out who we had to drag kicking and screaming into a "team environment" and who would just get on with it.
• I think building straw castles was a fun activity, however for me, it just made it apparent as to which people are more extraverted and willing to speak before people they don't know. It wasn't until about a month into the meetings did I realize that the quiet people have ideas too.
• Yes. It helped me realize who in the [team] had a "leader" mentality and who was more of a follower. Also, I was amazed how people can panic under pressure!
• We did not get to build castles. I think these problems would be good because it lets people see who will take charge etc.... and sort of establishes how people will work in the team.
• Yes I did. It showed us that we shouldn't be very unrealistic and unreasonably ambitious for our project. It was also fun that during early weeks in the term we got to know the team members more closely.
• Yes I learned how to interact with other members, how to make decisions.
• Not a lot, but there's still something I enjoyed...for example, to get to know my members' way of thinking and the way they handle problems.
• Opens flows of communications.
• Yes, it started the "breakdown" of the wall between working individually and working in a group. It helped get the members comfortable with each other and become more friendly. It was also a morale boost so as to enhance the motivation of the group and the performance of each team member.
• Yes it did help a lot such as communication, getting to know each other, working as a team.
• Yes, I really did. I realized to achieve a task involving others' participation I have to overcome first the communication part. Members should be clear, sure of task goal and their role in the course of task completion. Timing is very important too. To make decisions by myself is quick but with others is SLOW! Effective communication is essentially crucial.
• Yes helped us feel comfortable with sharing ideas, and let us get to know each other better.
• Yes they gave us a chance to get to know each other and do the group bonding thing.
• Yes, those events got the ball rolling, but talking during the meetings and after – simply getting to know the "real" people – has helped more.
• They were fun, but they didn't really help us feel like a team as much as the weekly meetings and working on the proposal.
• It was fun. So don't cancel that section.
• Not really, but it's fun.
• I do not believe that the number of activities held were necessary. It was good to have a couple, but I feel that some of the time could have been better spent orienting us to the unique structure and environment of 319. However, I did learn from the straw-castle exercise that planning your work is as important as doing it.
They were kinda fun, but I can't say they provided any real learning value to me.

Yes, but not enough to consider spending that much time on it. Would rather have started proposal at that time.

Personally, they made no difference. I do not believe that all of them were necessary. Maybe one kind of activity would be okay but no need for anymore.

No, not really. But it was a nice way to "break the ice."

I think that the building activities were good. It did break some of the ice although I heard people complaining that it's a waste of time.

The main benefit that I got out of the first several meetings was to get to know my teammates and my T.A. Nothing else seemed very relevant.

Didn't build castles but I felt we started a good rapport.

I enjoyed the "get-to-know-everyone" games.

Icebreakers were useful for building team confidence/associativity/familiarity.

Yes it is a good idea to have an ice-breaker.

Very good, helped to "break the ice" between team members who generally didn't know each other well.

Yes those activities broke the ice between team members that I did not know.

I missed that meeting due to a room change I didn't know about. However, I've done programs like that and I've felt it to be beneficial as it helps people become accustomed to one another, which is one of the hardest things to do.

I suppose so, you need to break the ice somehow.

Ummm. Yes, but only as an icebreaker.

I suppose they were vaguely useful for getting to know the rest of the team.

Yes I got to know how to deal with some of the team members better.

Yes it was a good exercise. Got familiar with other members.

Yes this activity draws members more together.

Yes, they let you get to know one another.

It was good in a way to introduce myself and to get to know others.

I think that we benefit in the way that we got to know each other, like an icebreaker. As far as whether these activities helped us learn about working in teams, I'm not too sure.

Introducing to each other certainly helps to know each other. But building a straw castle doesn't help much.

The straw building was fun. We got to play and joke around. But I don't know if it helped.

Hmmm... not really, we were just messing around.

The straw building activity was one activity that seemed useless, however the other activities did contribute in helping me know my team members better.

No, I am not into that sort of thing.

I am not able to see any direct benefit to the straw castle exercise.

No. I thought those activities were nonsense.

No, they were...er...interesting (not quite fun), but that's about it. I found the first couple weeks a waste of time. Perhaps it would be better [to] actually teach us how to be an effective group. COMM 329 is an Org. Behaviour class, taking some of the course's topics and presenting them at the beginning of the year would be very beneficial.
General Comments

COMMENTS RELATING TO THE WORKLOAD:

- Too much work for 3 credits. Should be a 6-credit course for the effort required.
- This course has to be 6-credits.
- 100 hours is more than I spend on an average 3-credit course. I am not looking forward to the amount of time this course will require next term.
- Would work better if this was a 3 or 6 credit 1-term course; we could assign more time to it. Teamwork is just as hard as regular work.
- This course should be worth 6 credits for the amount of work planned.
- I think that CS 319 should be a 6-credit course considering the amount of work we do: 1st term: 1 lecture, 1 meeting, assignments → ~ 4 hours a week. 2nd term: 800 hour project allocation, lectures, meetings → ~ 8 hours a week.
- I sometimes feel that this team stuff is very competitive. Teams outdoing each other could easily work well over their time allotted. I think that the time a person allots for this course should be stressed more.
- I try VERY hard to enjoy this course, but to be frank...I HATE it. It has nothing to do with T.A.'s, profs, my group members.... But a full UBC course load does not allow a person to enjoy the CS 319 environment (i.e. groups, proposals, teams, etc.). I think if CS 319 was worth 6 credits, so our overall course load is lighter it would be better.

CONCERNS RELATING TO HOW THE WORKLOAD IS DISTRIBUTED:

- Peter [the instructor] often stresses the fact that this is a 3-credit course and we don’t do any work this term but we do 100 hours next term. This is not true. It is also not good that he “deceives” us this way. There is a lot of work this term in terms of management. The group is simply too big.
- To have to specifically allocate every last hour is tedious. In real companies, projects are undertaken with a timeframe in mind, but if work has to get done, team members will keep working until each item is accomplished. Not until they’ve finished the ‘6’ hours allotted to it. Similarly here, our timeframe is roughly 100 hours, but allocating each hour is a bit too controlling.
- I heard from people who took 319 last year that they actually started coding well in advance. For this reason I am starting to worry about the work load of next term! Fortunately we are only doing one part of the ARS. I don’t know whether this is the idea of the ‘revised’ 319 or not. But this makes us being too lazy in the 1st term!!
- The structure of the course should be developed so that teams have an easier time allocating work. Parts of it don’t allocate at all! Just giving the RULES would help.

ON THE PROCESS OF GETTING STUDENT FEEDBACK:

- I think it is very important that a course get feedback like this from the students, especially when the course is a project. (Please voice this to the Dept.) The best thing about this course is that it’s open enough for the students to learn as much as s/he can/wants to. No limits. Lots of research. The fallback is obviously those who don’t do any work. I think the drastic change of the way 319 has been is very effective. It’s a great success.
- I think this questionnaire was helpful and that we should HAVE MORE of them. A greater class participation would make classes more interesting. Overall, I LIKE THIS COURSE.

OTHER IDEAS ON IMPROVING THE COURSE:

- I think the team leaders should be T.A.’s so we will first experience how the teams are supposed to work.
- Not enough time. I think this course should be taken apart into different courses. The first one should deal with a very simple design and concentrate more on the teamwork aspect. And the second part should emphasize a big project without the concentration on making group members work’ with each other.

The Importance of Team Skills for Software Development – by Carolyn Wick
GENERAL IMPRESSIONS:

- Good experience.
- I like the course, the team, and what my team is going to make.
- It makes the class more exciting – more than all the other CS classes – because there is more interaction, and the situation is more "realistic."
- This is a good course. I can learn how to develop software and more importantly how to interact with other members.
- It's very exciting, different from other courses.
- Taking CS 319, I learn how to cooperate with a group of people and I have a chance to talk in team meetings.
- I think this course should be a good experience of real world situations. But I hope you consider that it is our first experience and mark us considering that please.
- Interesting course. I like having the same team for the whole year. I hear how important this course's material is so looking forward to next term.
- The T.A.'s have been great at responding to questions in this course – this is a major plus for CS 319. (i.e. 11:30 responses at night! And coherent at that!)
- I'm aware that this is the first year [the course] was structured this way; and I understand there will be some problems. But I do not enjoy being a Guinea Pig.
- I do not like this course. Pointless IMHO. I have gained all these skills through work and I do not need a course to teach them to me.
- I like this course.
- 319 is one of the best courses in CPSC.
- Great course, glad I waited!
- Thanks!
Appendix III: Training Resources

The following materials were developed for teaching team skills within CPSC 319, and are included as sample resources for use with any undergraduate-level team project course or by any task-oriented team:

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>WORKSHEET OR HANDOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For Teaching Interaction Skills:</strong></td>
<td>Figure 12 Team Role Identification Worksheet (Part I)</td>
</tr>
<tr>
<td>These worksheets and reference sheets are to accompany a video-analysis of the team at work, or may be used (less effectively) by an individual process-observer during a team meeting. Adapted from Cascades Teaming Activity [Anderson Consulting, 1994], Marcic [1990, pp. 380-381], Scholtes [1988, pp. 42-43] and [Tubbs, 1988, p. 238].</td>
<td>Figure 13 Team Role Identification Worksheet (Part II)</td>
</tr>
<tr>
<td></td>
<td>Figure 14 Interaction Skills Examples (Part I)</td>
</tr>
<tr>
<td></td>
<td>Figure 15 Interaction Skills Examples (Part II)</td>
</tr>
<tr>
<td><strong>For Teaching Meeting Skills:</strong></td>
<td>Figure 16 Meeting Process Evaluation Form 1</td>
</tr>
<tr>
<td>These three evaluation forms were handed out to students to use in their meetings. Such process evaluation forms encourage open discussion of process and procedures, encourage monitoring the Meeting Basics, and help members find ways to improve the functioning of the team.</td>
<td>Figure 17 Meeting Process Evaluation Form 2</td>
</tr>
<tr>
<td></td>
<td>Figure 18 Meeting Process Evaluation Form 3</td>
</tr>
<tr>
<td><strong>For Developing Process Awareness:</strong></td>
<td>Figure 19 Personality Types Questionnaire</td>
</tr>
<tr>
<td>To draw students' attention to process, I had each of them complete this personality questionnaire. After students identified their own preferences, I led a discussion of how personality can affect group process.</td>
<td>Figure 20 Personality Questionnaire Scoring Instructions</td>
</tr>
<tr>
<td></td>
<td>Figure 21 Explanation of Personality Types</td>
</tr>
<tr>
<td><strong>For Stimulating Discussion and Awareness of Team Skills:</strong></td>
<td>Figure 22 Team Skills Questionnaire Page 1</td>
</tr>
<tr>
<td>This questionnaire was distributed to computer science graduate students, faculty, and software professionals from industry.</td>
<td>Figure 23 Team Skills Questionnaire Page 2</td>
</tr>
<tr>
<td>Team Members</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
</tr>
</tbody>
</table>

**Task-Related Constructive Roles**
1. Initiator
2. Information or Opinion Seeker
3. Information Giver
4. Opinion Giver
5. Clarifier
6. Summarizer
7. Consensus Seeker and Tester

**Process-Related Constructive Roles**
1. Planner
2. Feedback Giver or Receiver
3. Group Norms/Standards Setter and Tester

**Maintenance-Related Constructive Roles**
1. Energizer
2. Encourager
3. Gatekeeper
4. Compromiser
5. Surfer of Hidden Agendas or Concerns

**Destructive Self-Oriented Roles**
1. Storyteller or Gossiper
2. Dominator or Interruptor
3. Avoider, Withdrawer or Abandoner
4. Negativist

---

**Figure 12. Team Role Identification Worksheet (Part I)**

_The Importance of Team Skills for Software Development – by Carolyn Wick_
1. What did your team do well? (Which constructive interaction roles did team members fulfill?)

2. In what areas could your team improve? (Which types of interaction could your team benefit from increasing, and which roles— if any— could your team benefit from decreasing?)

3. What did you personally do well? (Assess the roles you played on the team and discuss them with the team. Ask other team members for feedback on roles you played.)

4. In what areas could you personally improve to help this team?

Figure 13. Team Role Identification Worksheet (Part II)
**TASK-RELATED CONSTRUCTIVE ROLES:**

<table>
<thead>
<tr>
<th>Interaction Function</th>
<th>Examples</th>
</tr>
</thead>
</table>
| 1. Initiating                               | • "One of the things we have to do is ... because .... Why don't we begin by ...."  
• "Has anyone had a chance to think about what sort of on-line help we want to implement?"  
• "Let's start brainstorming ideas! What could we name our product?" |
| 2. Information and opinion giving           | • "It says here in this requirements specification that there should be three types of queries instead of two; we seem to have forgotten about ...."  
• "The customer never explicitly requested that feature. Maybe we should cut it out of our first version?"  
• "What if we ...." "I think we should...." |
| 3. Information and opinion seeking          | • "Can someone summarize the main considerations discussed so far?"  
• "Let's go around the table and hear everyone's ideas."  
• "I'm confused. Could someone please explain why we decided so quickly not to do such-and-such?"  
• "Does anyone have any remaining points to make before we move on to the next agenda item?" |
| 4. Coordinating and summarizing             | • "What if we used Bob's 3D scrollbar idea in Jan's browser window?"  
• "We seem to have come up with three possible approaches...." |
| 5. Clarifying and elaborating               | • "Dave has a point. <Restate/reword points>"  
• "That idea sounds a lot like what Jenny just mentioned. Jenny, could you elaborate on what you had in mind?"  
• "If I understand you correctly, you're suggesting that we.... Is that right?"  
• "How is this idea different from....?"  
• "Maybe we can't do such-and-such per se, but perhaps we could...." |
| 6. Evaluating                               | • "One advantage/disadvantage of such-and-such might be...."  
• Avoid: "That will never work." "That's a ridiculous idea!" "That's the best idea yet!" "We already tried that!" "How could you do that when you knew/I told you....??!"  
• Try writing all suggestions on a board visible to the entire group, thereby separating ideas from persons, then as a group objectively pull out the best ideas. |
| 7. Consensus testing and terminating         | • "How many people are into developing a UNIX version?"  
• "So it sounds like we need a month. Would February 14th be okay for everybody?" |

Figure 14. Interaction Skills Examples (Part I)
**PROCESS-RELATED CONSTRUCTIVE ROLES:**

<table>
<thead>
<tr>
<th>Interaction Function</th>
<th>Examples</th>
</tr>
</thead>
</table>
| 1. Structuring and planning | • “To save time, how about if we break into subgroups to explore each of these three remaining alternatives and then report back to the whole group?”
• “We’ve gone a bit off topic and we’re running out of time, but it sounds like this is an important issue to resolve. What if we move such-and-such an item to the next meeting’s agenda so we can have another 20 minutes to discuss this now?” |
| 2. Giving and receiving feedback | • “I’d appreciate comments about the way I managed the time today. Did you feel rushed?”
• “When you ..., I felt ...
• “I would like it if you could ... because ...” |
| 3. Processing and standard setting | • “Hey, it seems that every time someone makes a suggestion, the idea gets shot down right away. Maybe we should hold off judging until we’ve collected all our alternatives?”
• “What can we do so this doesn’t happen next time?”
• “We spent 3 days implementing that, although only 2 hours were allocated to it.” |

**MAINTENANCE-RELATED CONSTRUCTIVE ROLES:**

<table>
<thead>
<tr>
<th>Interaction Function</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Energizing</td>
<td>• Novel ideas, humour, a short break, food, a change of pace, some statement of feeling, suggesting some participatory activity to energize the group, and so on.</td>
</tr>
</tbody>
</table>
| 2. Encouraging | • Smile. Use names. Make eye contact.
• “Hey, that’s an interesting idea, Carla.” |
| 3. Gatekeeping | • “Were you going to say something, Jim?” |
| 4. Compromising | • “I’m sorry.”
• “Well, I guess we don’t have to do it the way I suggested as long as we come up with a plan that would take about the same amount of time.” |
| 5. Harmonizing, consensus seeking, and surfacing hidden agendas | • “What is it that makes you feel so strongly about such-and-such an idea? Would you be willing to explore some alternatives with me to see if we can find a new solution that meets your underlying criteria?”
• “It actually sounds like we’re talking about two distinct issues: ... and ...” |

**DESTRUCTIVE SELF-ORIENTED ROLES:**

<table>
<thead>
<tr>
<th>Destructive Function</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Storytelling or Gossiping</td>
<td>• Rambling, or going unnecessarily off topic, possibly for attention.</td>
</tr>
<tr>
<td>2. Dominating or Interrupting</td>
<td>• Restating a point too many times, without properly listening to others’ contributions.</td>
</tr>
<tr>
<td>3. Avoiding, Withdrawing or Abandoning</td>
<td>• “Whatever you say.”</td>
</tr>
<tr>
<td>4. Negativity</td>
<td>• Obstinate arguing or finding fault with things and griping.</td>
</tr>
</tbody>
</table>

Figure 15. Interaction Skills Examples (Part II)
**Meeting Basics**

*Directions:* Please evaluate the degree to which this meeting incorporated each of the Meeting Basics by circling the appropriate number on the scale to represent your rating.

<table>
<thead>
<tr>
<th>AGENDA:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>agenda distributed in advance, followed; items ordered by urgency, times allocated by importance</td>
</tr>
<tr>
<td>no apparent agenda, or agenda not followed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECTIVE:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>objective defined, understood, and met</td>
</tr>
<tr>
<td>no apparent objective</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FACILITATOR:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>facilitator designated in advance, kept discussion focussed on task, started and ended meeting on time</td>
</tr>
<tr>
<td>no designated facilitator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MINUTES:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>someone took minutes and can distribute them to participants; action items clearly identified</td>
</tr>
<tr>
<td>no one took minutes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NEXT MEETING:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>next meeting date, time, location, and facilitator set; objective and agenda drafted</td>
</tr>
<tr>
<td>next meeting was not mentioned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVALUATION:</th>
<th>1 2 3 4 5 6 7</th>
<th>perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>poor</td>
<td></td>
<td>a format existed for the facilitator to receive/give feedback, and for participants to comment specifically on what went well, what to improve</td>
</tr>
<tr>
<td>no time was spent formally reflecting on process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Formative Evaluation

Directions: Please give your candid reactions to this meeting by rating it on the seven-point scale (circle the appropriate number on the scale to represent your evaluation) and by responding to the questions below. Your feedback is appreciated.

1. What did you think of this meeting?
   
   A waste of time 1 2 3 4 5 6 7 Much was accomplished

2. What went well? Be specific.

3. What could have been done differently to improve the meeting? List suggestions.

4. What could the facilitator have done differently? List suggestions.

5. What might you have done to help the meeting?
The One-Minute Paper*

Please answer each of the following questions in 1 or 2 sentences:

1. In your opinion, what was the most useful/meaningful thing that we accomplished or that you learned during this session?

2. What question(s), issues, or concerns remain uppermost in your mind as we end this session?

*Adapted from Cross & Angelo, Classroom Assessment Techniques, 1988, pp. 148-150.
U.C.Berkeley Classroom Research Project (6/90)

Figure 18. Meeting Process Evaluation Form 3
Instant Insight Inventory

either A or B for each statement.

A I answer a question quickly, sometimes without thinking. or 1. B I like to think about something before I offer an answer or an opinion.

A I use trial and error with confidence. or 2. B I like to go deeply into understanding something before I try it.

A I need to find out what others expect of me. or 3. B I like to do things on my own.

A I get full of energy when I am around a lot of people, such as at a party. or 4. B I get tired when I am around a large group of people, and need to get away often to be by myself and collect my thoughts.

A I enjoy a lot of variety and action. or 5. B I enjoy a quiet place all my own where I can reflect uninterrupted.

A I enjoy looking at details and proof that things are really as they appear to be. or 6. B I tend to skim over details and look for hidden meanings in things.

A I enjoy checking, inspecting, reading the fine print, finding out all the information I can. or 7. B I become impatient with routine and repetitive slow, precise activities.

A I enjoy things as they are, recall past events, and learn from the combination of these two in a “common sense” sort of way. or 8. B In a flash of insight “I go with my hunches” on many things.

A It would be fairly accurate to describe me as being realistic and practical. or 9. B It would be fairly accurate to describe me as being imaginative and inventive.

A I rarely rely on inspiration to keep me going. or 10. B I have a lot of bursts of energy, with slack periods in between.

A I need for justice rules much of what I do. or 11. B Harmony is one of the most important aspects of my life.

A I try to analyze logically all the facts in making a decision. or 12. B I think of what is best for all the people involved in making a decision.

A I consider fair and honest criticism to be a natural, acceptable part of human relationships. or 13. B I avoid confrontation and feel very uncomfortable giving or receiving criticism.

A I know lots of people who are too soft-hearted and emotional in making decisions. or 14. B I have my feelings hurt by people who tend to analyze or make cold statements when understanding is what I am looking for.

A I often have difficulty in freely expressing the emotions I am experiencing. or 15. B I find it easy to express my feelings and to understand the feelings of others.

A I like to be in control of the events in my life and make them “the way they ought to be.” or 16. B I need to understand thoroughly the events in my life and therefore spend more time than I should in making decisions.

A I decide things slowly and change my mind often. or 17. B I decide things slowly and change my mind often.

A I like schedules and some definite order or system to regulate the way I do things. or 18. B I prefer an easy-going flexible pattern to live by.

A I choose work to come before play in my organization of time and priorities. or 19. B I allow my “there is plenty of time” attitude to make meeting deadlines a rather mad-rush affair for me.

A I enjoy most friends who share my ideals and standards and are true to them. or 20. B I choose friends who have interests similar to mine and with whom I can share common experiences.

This material taken from Simon Fraser University’s Student Leader Training Resources (original source unknown).

Figure 19. Personality Types Questionnaire

The Importance of Team Skills for Software Development – by Carolyn Wick
To determine which MBTI personality type most closely describes the preferences you marked on the questionnaire, add up the number of A’s and B’s circled for the following sets of questions:

<table>
<thead>
<tr>
<th>Questions 1-5:</th>
<th>A’s ___</th>
<th>B’s ___</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you have more A’s than B’s, put an “E” in the box.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you have more B’s than A’s, put an “I” in the box.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions 6-10:</th>
<th>A’s ___</th>
<th>B’s ___</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you have more A’s than B’s, put an “S” in the box.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you have more B’s than A’s, put an “N” in the box.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions 11-15:</th>
<th>A’s ___</th>
<th>B’s ___</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you have more A’s than B’s, put a “T” in the box.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you have more B’s than A’s, put an “F” in the box.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions 16-20:</th>
<th>A’s ___</th>
<th>B’s ___</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you have more A’s than B’s, put a “J” in the box.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you have more B’s than A’s, put a “P” in the box.</td>
<td></td>
</tr>
</tbody>
</table>
### Characteristics frequently associated with each type

<table>
<thead>
<tr>
<th>Sensing Types</th>
<th>Intuitive Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ISTJ</strong></td>
<td><strong>INFJ</strong></td>
</tr>
<tr>
<td>Serious, quiet, earn success by concentration and thoroughness. Practical, orderly, matter-of-fact, logical, realistic, and dependable. Take responsibility. Make up their own minds as to what should be accomplished and work toward it steadily, regardless of protests or distractions.</td>
<td>Smart by perseverance, originality, and desire to do whatever is needed or wanted. Full their best efforts into their work. Quiet, thoughtful, concentrated, concerned for others. Respectful for their firm principles. Likely to be honored and followed for their clear convictions as to how best to serve the common good.</td>
</tr>
<tr>
<td><strong>ISFJ</strong></td>
<td><strong>INTJ</strong></td>
</tr>
<tr>
<td>Quick, friendly, responsible, and conscientious. Work directly to meet their obligations. Indebted to any project or group. Through planning, precision, accuracy. Their interests are usually not technical. Can be patient with necessary delays. Loyal, considerate, perceptive, concerned with how other people feel.</td>
<td>Usually have original minds and great drive for their own careers and purposes. In tasks that appeal to them, they have a fine power to organize a job and carry it through with or without help. Skeptical, critical, independent, determined, sometimes stubborn. Must learn to yield less important points in order to win the most important.</td>
</tr>
<tr>
<td><strong>ISTP</strong></td>
<td><strong>INFP</strong></td>
</tr>
<tr>
<td>Good observers—quiet, reserved, observing and analyzing life with detached curiosity and unexpected flashes of original humor. Usually interested in cause and effect, how and why mechanical things work, and in organizing facts using logical principles.</td>
<td>Full of enthusiasm and loyalty, but seldom talk of these until they know you well. Care about learning, ideas, language, and independent projects of their own. Tend to undertake too much, then somehow get it done. Friendly, but often too absorbed in what they are trying to be right. Little concerned with perfections or physical surroundings.</td>
</tr>
<tr>
<td><strong>ESFP</strong></td>
<td><strong>ENFP</strong></td>
</tr>
<tr>
<td>Good at on-the-spot problem solving. Do not worry, enjoy whatever comes along. Tend to like mechanical things and sports, with friends on the side. Adaptible, tolerant, generally conservative in values. Like clear explanations. Are best with real things that can be worked, handled, taken apart, or put together.</td>
<td>Warmly enthusiastic, high spirited, imaginative, versatile. Able to do almost anything that attracts them. Quick with a solution for any difficulty and ready to help anyone with a problem. Often rely on their ability to improvise instead of preparing in advance. Can usually find compromising reasons for whatever they need.</td>
</tr>
<tr>
<td><strong>ESFJ</strong></td>
<td><strong>ENTJ</strong></td>
</tr>
<tr>
<td>Warm-hearted, talkative, popular, conscientious, born cooperators, active committee members. Need harmony and may be good at creating it. Always doing something new in interview. Work well with minimum attention and pressure. Main interest is in things that directly and visibly affect people's lives.</td>
<td>Impetuous, quick, decisive, leaders in activities. Usually good in anything that requires reasoning and intelligent talk, such as public speaking. Are usually well advanced and enjoy adding to their stock of knowledge. May seem to oppose more positive and confident than their experience as an area warrants.</td>
</tr>
</tbody>
</table>

Figure 21. Explanation of Personality Types

This material taken from Simon Fraser University's Student Leader Training Resources (original source unknown).
How important are team skills for software engineers?

Directions: Please answer the following 5 questions and then drop the questionnaire into the box provided. Your comments will help determine how much emphasis should be placed on teaching non-technical skills and concepts to aspiring software engineers.

1. Meeting Skills: How many meetings do you attend in a typical week? __________

   What do you think of most of your meetings? (Circle the appropriate rating on the scale)

   A waste of time 1 2 3 4 5 6 7 Much was accomplished

2. Obstacles to Positive Team Experiences:

Which of the following symptoms of group dysfunction have you observed in the career-related groups to which you belong?

   never 1 2 3 4 5 6 7 frequently

   a. Apathy
   b. Poor leadership
   c. Inability to make decisions
   d. Hidden agendas
   e. Counter-productive conflict
   f. Poor communication skills

3. Team Skills Training: For each of the following skills, indicate how you think software engineers should learn the skill (you may select more than one source):

<table>
<thead>
<tr>
<th>Formal Education</th>
<th>Trial &amp; Error</th>
<th>Not an important skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>in University?</td>
<td>(life experience)</td>
<td></td>
</tr>
<tr>
<td>in Industry?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   a. How to run effective meetings.
   b. Leadership skills.
   c. Public speaking skills.
   d. Listening skills.
   e. Conflict management skills.
   f. Marketing, Negotiation skills.
   g. Letter and report writing skills.
   h. Ability to relate to different cultures (ethnic, gender).

Figure 22. Team Skills Questionnaire Page 1
How important are team skills for software engineers? (cont.)

4. Relative Importance of Team Skills vs. Technology Skills:
Which graph best shows the relative importance of team skills for software engineers (as compared with technology skills and expertise)? (circle one)

a.  
Team | Technology
--- | ---

b.  
Team | Technology
--- | ---

c.  
Team | Technology
--- | ---

d.  (draw your own)

5. Personal Data: Which one of the following best describes you in your current occupation?

- software developer
- software tester
- project manager
- human resources person
- student
- faculty/researcher
- other: ____________________

Would you like to participate further? If you leave your name and number, or attach a business card, you can participate in a follow-up survey or receive an update as this research progresses.

Name: _____________________ How to Contact Me: _____________________

Any Additional Comments?

___________________________________________________________

Thanks!

Figure 23. Team Skills Questionnaire Page 2