Preface

Software engineering is almost always a collaborative activity. This book brings together a number of recent contributions to the domain of Collaborative Software Engineering (CoSE) from a range of research groups and practitioners. These range from tools and techniques for managing discrete, low-level activities developers engage in when developing parts of software systems; knowledge, project and process management for large scale collaborative software engineering enterprises; and new ways of organizing software teams including outsourcing, open sourcing, highly distributed virtual teams and global software engineering. We believe that all practitioners engaging in or managing collaborative software engineering practices, researchers contributing to advancement of our understanding and support for collaborative software engineering, and students wishing to gain a deeper appreciation of the underpinning theories, issues and practices within this domain will benefit from most if not all of these contributions.

Introduction

Ever since people began to create software there has been a need for collaborative software engineering. At some point people need to share their code and designs with others. Software frequently grows large and complex, thus requiring a team of multi-talented experts to work together to tackle the project. Such a team must adopt suitable processes and project management to ensure the myriad of tasks are completed; to keep track of what each other is doing; and to ensure the project advances on-time, on-budget and with the software meeting appropriate quality levels. The team must share both low-level artifacts and higher-level knowledge in controlled, consistent ways, be proactively informed of changes others make, and co-ordinate their work “in the small” as well as “in the large”. Various studies have demonstrated that peer review of designs and code improve them, leading to collaborative testing and quality assurance practices. Recent trends have moved software across organizational and country boundaries, including virtual software teams and open source software development. Agile methods have brought bottom-up, human-oriented processes and techniques to bear that are very different from traditional, centralized and hierarchical development practices.
Our understanding of and support for collaborative software engineering has advanced tremendously over the past forty years. We understand that team formation and management is not a straightforward task. However we are still learning about formation, management and evolution in domains such as agile teams, projects with substantive outsourcing, open source software, virtual software teams and global software engineering domains. Knowledge management is critical in software engineering and we have developed as a community many approaches to representing knowledge about software as well as tools to facilitate its capture. However, shared, evolving knowledge and appropriate tools and techniques to support this is less well-understood from both theoretic and practical standpoints. How do we best represent and collaboratively manage knowledge about requirements, architecture, designs, quality assurance measures and software processes themselves? Social influences on software engineering and teams have become more important as have organizational implications. How do team members relate to one other and how to we build effective team relationships for communication, co-ordination and collaboration? How do we set up a successful multi-site software project? A successful open source project? A successful outsourcing project?

The actual act of collaborative software creation has received much attention over many years. But what are the right sets of tools and work practices to deploy on a collaborative software engineering project to best-support engineers and ensure quality? What are the unsolved issues around co-ordination especially in large or highly distributed teams? Configuration management remains one of the most challenging activities in collaborative software engineering.

**Book Overview**

We have divided this book into four parts, with a general editorial chapter providing a more detailed review of the domain of collaborative software engineering. We received a large number of submissions in response to our call for papers and invitations for this edited book from many leading research groups and well-known practitioners of leading collaborative software engineering techniques. After a rigorous review process 17 submissions were accepted for this publication. We begin by a review of the concept of collaborative software engineering including a brief review of its history, key fundamental challenges, conceptual models for reasoning about collaboration in software engineering, technical, social and managerial considerations, and define the main issues in collaborative software engineering.

Part I contains five chapters that characterize collaborative software engineering. This includes characterizing global software engineering via a process-centric approach, requirements-driven collaboration using requirements/people relationships, decoupling in collaborative software engineering, agile software development and co-ordination, communication and collaboration, and applying the concept of ontologies to collaborative software engineering.

Part II contains five chapters that examine various techniques and tool support issues in collaborative software engineering. This includes an analysis of
awareness support in collaborative software development teams, an overview of several approaches and tools to supporting continuous co-ordination, a maturity model for outsourcing offshore, an architectural knowledge management platform, and a set of design principles for collaborative software engineering environments.

Part III contains three chapters addressing the issue of organizational issues in collaborative software engineering. This includes supporting the concept of collaborative software analysis and making analysis tools widely accessible, open source software project communication and collaboration analysis and visualization support, and a review and critique of multi-site software development practices.

Part IV contains four chapters looking at a variety of related issues in the collaborative software engineering domain. These include key open source/free software development collaboration issues, configuration management and collaborative development, knowledge sharing to support collaborative software architecting, and rationale management to enhance collaborative requirements engineering. We conclude with a summary of current challenges and future directions in collaborative software engineering.

What Is Collaborative Software Engineering?

Collaboration has been a necessity ever since software engineering began. The early days of software engineering saw very limited process, technique and tool support for collaboration. Early efforts to support collaboration were limited to structured, waterfall-based processes, early version control tools, rigid team role specialization, and centralization of software activities. The advent of Computer-Aided Software Engineering tools and Integrated Development Environments introduced a wider, more accessible range of collaboration support mechanisms including awareness support, collaborative analysis and reviews and iterative, rapid applications development processes. More recently has seen the growth of distributed teams, outsourcing, open source software projects, global software engineering processes and highly decentralized team support tools.

Fundamental challenges in collaborative software engineering remain the same: the need to share artifacts, communicate and coordinate work. These occur across a spectrum of low-level to high-level. Low-level challenges include making shared artifacts like code, tests and designs accessible in a timely manner to team members while controlling access, ownership, integrity and quality. Large software projects require effective version control and configuration management techniques and tools. Knowledge management is fundamental especially around design rationale, architecture and processes. Software development has changed dramatically over the past 10 years. This is evidenced by new organizational and team dynamics including open source software, software outsourcing, distributed teams, and global software engineering. Choice of processes, project management, tools and evolution of software in these domains is still an emerging field of research and practice.

Key technical considerations in collaborative software engineering revolve around process, project management, knowledge and configuration management
and tool platform selection and operation. A software process and project management regime must be chosen that supports collaboration appropriate to the team, project and organizational circumstances. These range from small, single-site/single-project teams, to large team/multi-project/multi-site domains. The later may include outsourcing and open source components. Complex software systems require effective knowledge management approaches and support tools. They also require scalable configuration management tools. Tool platforms and collaboration-supporting components have become very diverse. These range from small-team, homogeneous IDEs with awareness and collaboration plug-ins to highly diverse platforms where software engineering is part of a larger systems engineering activity. Communication support between engineers often becomes a crucial component of the team support infrastructure.

Being an inter-personal and—often—inter-organizational activity, collaborative software engineering introduces a number of social and managerial challenges. Teams may be homogeneous or highly diverse in terms of culture, language and location. This introduces many challenges to supporting collaboration at high levels (process, project management) and low-levels (artifact sharing, consistency). Teams may be comprised of many generalist’s e.g., agile methods or highly specialized individuals or sub-teams whose efforts must be coordinated. An organization needs to ensure appropriate management of teams and between teams. In particular, global software engineering domains introduce very new and challenging problems, such as in contracting and quality control in outsourcing, ownership and “group dynamics” in multi-site projects, and overall project direction and co-ordination in open source software projects.

Part I – Characterizing Collaborative Software Engineering

The five papers in this section identify a range of themes around the characteristics of collaborative software engineering. There has been a dramatic increase in interest in the concept of “global software engineering” over the past 10 years. This has included the increasing number of distributed, multi-site software engineering teams; outsourcing of software engineering activities, often in search of cost savings and capacity limits, and open source software development. Each of these trends brings with it added complexity to the engineering process—software engineers are no longer co-located, are no longer in regular face-to-face contact (if at all), and different time zones, cultures and languages enter the mix.

A number of studies have been undertaken to better-understand the issues of collaboration challenges in such “virtual” software team environments. A key aim is to understand factors that adversely impact on collaboration practices and factors that support communication, co-ordination and collaboration in such domains. Studies have focuses on a range of organizations, projects and team sizes. One area of particularly detailed study has been requirements engineering. A distributed team develops and shares a set of requirements and a crucial factor impacting quality of these is communication strategies.
Preface

Knowledge engineering has become important in collaborative software engineering. One aspect is the development of ontologies, or shared semantic meanings, of software artifacts and processes. These enable co-ordination of activities along with improved communication about shared concepts in domains ranging from requirements engineering to software architecture.

Agile methods have become popular in many domains of software engineering. A characteristic is their focus on people-centric aspects of software engineering tasks, including communication and co-ordination. Pairing is one aspect of several agile methods that offers a tangible way to encourage improved collaboration outcomes.

Part II – Tools and Techniques

Software engineering requires a number of complex, interleaved activities to be carried out. These must be organized into logically correct teamwork and be supported by appropriate tools. Because of the challenges of supporting collaborating in an already complex engineering process, a multitude of techniques and tools have been developed to support almost all activities of collaborative software engineering.

Traditionally software engineering had been a co-located activity where team members could expect some degree of face-to-face communication and co-ordination were important activities but discrete and compartmentalized. Outsourcing parts of a software engineering project and highly iterative agile processes have led to an increased interest in how to best support virtual, distributed collaboration and communication and co-ordination for team activities that repeat in days rather than months.

A range of support mechanisms and associated tool support have appeared in recent years to address concerns in both traditional but more particularly these newer domains of collaborative software engineering. Social networking-style support such as tagging, shared knowledge repositories and communication support have become popular. New search-based support and associated visualization support have become more important as developers are less familiar with large tracts of software systems. These include mining of software repositories and context-aware filtering mechanisms in IDEs. Event-based support mechanisms have always been popular in collaborative support environments. These have been explored further in the context of both same-place and distance-located teams to support proactive notification and various levels of group awareness.

Developer-centric software engineering tools are crucial and this includes support for collaboration. Areas of particular interest in these tools are knowledge management and expertise communication. Knowledge management requires use of shared ontologies and supporting authoring tools, but as importantly the development of true “virtual communities” where informal knowledge sharing is supported and encouraged. Expertise communication is one aspect where the collaboration environment allows increasingly geographically dispersed team members to better communicate both knowledge and expertise relating to knowledge and tasks.
Part III – Organizational Experiences

Multi-site, or geographically distributed software development, has introduced a range of unknowns into software engineering practice and research. Of particular note is the lack of guidance around process selection. When running a multi-site, geographically distributed software project, what is the “best” software process to choose to organize this activity, quite apart from tool, project management and team selection issues? How can organizations make process choices, in particular, to best exploit multiple time zones, team expertise, out-sourced and open-sourced parts of a product, and ensure quality, cost and timeliness thresholds? Two fundamental ways of organizing a distributed project are centralized control of overall process and distribution of scoped design/code/test, compared with distributing different phases e.g. requirements team, design and build team, testing team in different locations.

Open source software projects are an increasingly common model of distributed, virtual software teams. Many studies have looked at collaboration aspects of such projects, in particular the evolution of the code base and team communication and co-ordination patterns. Recovering such information is challenging—often via bug reports, detailed code analysis and informal interviews of key team members. It is still an unsolved research problem how to best set up an open source project to achieve high quality communication and co-ordination.

Software artifact analysis has been used extensively for many years. This includes static analysis of source code, tests, designs and requirements and dynamic analysis of execution traces, side-effects and formal models of code. Collaboration around analysis has often been informal and poorly structured. Given the increasing complexity of code and analysis tools and techniques, an open challenge is how to share analysis processes and techniques, and also the tools supporting these, particularly across organizations.

Part IV – Related Issues

A number of socio-technical issues arise in collaborative software engineering. In free and open source software development projects these are particularly challenging. Key issues include overall project ownership and co-ordination, task de-composition, trust, accountability, commitment and social networking. Collaboration affordances in the individual and group development ecosystem must support both the range of collaboration activities but take into account the free and open source domain of work.

Knowledge sharing is crucial in all domains of software engineering. Particular domains of interest include requirements engineering and software architecture where commissioner, engineer, manager and end user constraints intersect and often must be balanced. Knowledge sharing in collaborative software architecting supports better decision making, surfacing of assumptions, and reasoning about design decisions. In product line engineering, variability management is a key challenge,
particularly when faced with multi-site software teams. Rationale management can be used to augment the variability management process to improve collaboration support in this context.

Configuration management has long been a challenge in software engineering particularly as systems have grown enormously in size and complexity. As configuration management requires integrating many software artifacts and ultimately impacts all phases of proceeding development, configuration management support systems have been an early contributor to collaborative software engineering infrastructure. They provide a shared space, awareness support, record and enable tracing of team actions, and support both knowledge sharing and communication. Many outstanding research and practice issues exist in each of these areas of configuration management systems support, however, leading to next generation collaborative software engineering tools.

**Current Challenges and Future Directions**

Collaborative software engineering has been a very heavily researched area and almost all practicing software teams will need to engage in it. However, many challenges still present both in terms of adopting collaboration practices, processes and tools and improving the state-of-the-art. Many of these challenges are long standing, and hence are fundamental to the act of working together to engineer shared artifacts. These include assembling teams, dividing work, social networking within and between teams, choosing best-practice processes, techniques and supporting tools, and effective project management. Others have arisen due to new organizational practices and technical advances, including open-sourced, out-sourced, multi-site and agile software engineering contexts. We still do not know the ideal way to share knowledge, facilitate the most effective communication, co-ordinate massively distributed work, and design and deploy support tools for these activities.

Auckland, New Zealand John Grundy
Heidelberg, Germany Ivan Mistrík
Irvine, CA, USA André van der Hoek
Santa Cruz, CA, USA Jim Whitehead
Collaborative Software Engineering
Mistrik, I.; Grundy, J.; van der Hoek, A.; Whitehead, J. (Eds.)
2010, XXIV, 409 p., Hardcover
ISBN: 978-3-642-10293-6