Sungdeok Cha · Richard N. Taylor
Kyochul Kang *Editors*

Handbook of Software Engineering

Springer

Handbook of Software Engineering

Sungdeok Cha • Richard N. Taylor • Kyochul Kang
Editors

Handbook of Software Engineering

Springer

Editors
Sungdeok Cha
College of Informatics
Korea University
Seoul, Korea (Republic of)

Richard N. Taylor
University of California, Irvine
CA, USA

Kyochul Kang
Professor Emeritus
POSTECH
Pohang, Korea (Republic of)

ISBN 978-3-030-00261-9 ISBN 978-3-030-00262-6 (eBook)
https://doi.org/10.1007/978-3-030-00262-6

Library of Congress Control Number: 2018960866

Preface

2019—the year of this handbook's publication—marks the 50th anniversary of the traditional birth of the discipline of software engineering. Now with substantial maturity, software engineering has evolved from narrow concerns in "coding" to cover a broad spectrum of core areas, while mingling at the edges with the disciplines of human–computer interaction, programming language design, networking, theory, and more. This volume provides a concise and authoritative survey of the state of the art in software engineering, delineating its key aspects and serving as a guide for practitioners and researchers alike.

This handbook is unique among other handbooks (e.g., the Software Engineering Body of Knowledge or SWEBOK) in several aspects.

First, each chapter provides an organized tour of a critical subject in software engineering. The central concepts and terminology of each subject are laid out and their development is traced from the seminal works in the field. Critical readings for those seeking deeper understanding are highlighted. Relationships between key concepts are discussed and the current state of the art made plain. These presentations are structured to meet the needs of those new to the topic as well as to the expert.

Second, each chapter includes an in-depth discussion of some of the field's most important and challenging research issues. Chosen by the respective subject matter experts, these topics are critical emphases, open problems whose solutions may require work over the next 10–15 years.

Articles in the handbook are appropriate to serve as readings for graduate-level classes on software engineering. Just as well, chapters that describe some of the most fundamental aspects of software development (e.g., software processes, requirements engineering, software architecture and design, software testing) could be selectively used in undergraduate software engineering classes.

A distinguishing characteristic of this volume is that in addition to "classical" software engineering topics, emerging and interdisciplinary topics in software engineering are included. Examples include coordination technologies, self-adaptive systems, security and software engineering, and software engineering in the cloud.

Software engineering practitioners in the field can thus get a quick but in-depth introduction to some of the most important topics in software engineering, as well as topics of emerging importance. Selective references at the end of each chapter guide readers to papers to obtain more detailed coverage on specific concepts and techniques.

No handbook can be considered complete nor will it remain relevant indefinitely due to rapid advances in software engineering technologies. Some topics are omitted here because, while having deep roots in software engineering, due to their maturity they are no longer broadly active research areas. Configuration management is an example. Some topics are, unfortunately, left out because of practical constraints. The editors believe that this handbook will best serve the community of software engineering researchers and practitioners alike if it is updated regularly.

Enjoy reading the 2019 state-of-the-art survey in software engineering presented by respected authorities in each of the subject areas. Needless to say, contributors to various chapters deserve the most credit for their generosity to share their expertise with the community and donate their precious time.

Seoul, Korea	Sungdeok Cha
Irvine, CA, USA	Richard N. Taylor
Pohang, Korea	Kyochul Kang

Acknowledgment

The editors of this handbook would like to thank the authors who spent many hours of their highly demanding schedule in writing the manuscripts. Without their enthusiastic support, this book would not exist. The Springer publication team, especially Ralf Gerstner, deserves a special note of appreciation for support and patience during the period when progress on this handbook project slowed.

Several people helped the editors as anonymous reviewers of various chapters, and we acknowledge their contribution. Special thanks are due, not in particular order, to: Kenji Tei (National Institute of Informatics, Japan), Kyungmin Bae (POSTECH, Korea), Moozoo Kim (KAIST, Korea), and Jaejoon Lee (Lancaster University, UK).

Finally, the editors would like to express personal acknowledgment for the support they received while working on this book project.

Sungdeok (Steve) Cha thanks his wife, Christine Yoondeok Cha, for her endless love and support in prayer. This book is my gift to you, Yoondeok. Richard Taylor thanks his wife, Lily May Taylor, for her continuing support, in this year of our 40th wedding anniversary. Kyochul Kang thanks the late Prof. Daniel Teichroew for leading him into software engineering research. Also, sincere gratitude goes to his wife Soonok Park for her loving care and support whenever he challenged for a new career.

Contents

Editors and Contributors

About the Editors

Sungdeok Cha is a Professor at Korea University in Seoul, Korea and a former professor at Korea Advanced Institute of Science and Technology (KAIST) in Daejeon. Prior to joining KAIST, he was a member of technical staff at the Aerospace Corporation and the Hughes Aircraft Company working on various software engineering and computer security projects. His main research topics include software safety, requirements engineering, and computer security. He is also a member of editorial boards for several software engineering journals.

Richard N. Taylor is a Professor Emeritus of Information and Computer Sciences at the University of California, Irvine, USA. His research interests are centered on design and software architectures, especially focusing on decentralized systems. In 2017 he received the ACM SIGSOFT Impact Paper Award (with Roy Fielding). In 2009 he was recognized with the ACM SIGSOFT Outstanding Research Award, in 2008 the ICSE Most Influential Paper award, and in 2005 the ACM SIGSOFT Distinguished Service Award. In 1998 he was named an ACM Fellow for his contributions to research in software engineering and software environments.

Kyochul Kang is an Executive Vice President at Samsung Electronics as well as a Professor Emeritus at POSTECH in Korea. Prior to joining POSTECH, he conducted software engineering research at Bell Communications Research, Bell Labs, and SEI. His research career in software engineering began in the 1970s as a member of the PSL/PSA team, which developed the first-ever requirements modelling and analysis technology. He is well known for his FODA (Feature-Oriented Domain Analysis) work at SEI and is an expert on software reuse and product line engineering.

Contributors

Eduardo Santana de Almeida Federal University of Bahia, Salvador, Bahia, Brazil

Hamid Bagheri University of Nebraska-Lincoln, Lincoln, NE, USA

Amel Bennaceur The Open University, Milton Keynes, UK

Eric M. Dashofy The Aerospace Corporation, El Segundo, CA, USA

Gordon Fraser University of Passau, Passau, Germany

Joshua Garcia University of California, Irvine, CA, USA

Volker Gruhn Lehrstuhl für Software Engineering, Universität Duisburg-Essen, Essen, Germany

Yann-Gaël Guéhéneuc Polytechnque Montréal and Concordia University, Montreal, QC, Canada

Foutse Khomh Polytechnque Montréal, Montreal, QC, Canada

Miryung Kim University of California, Los Angeles, CA, USA

Sam Malek University of California, Irvine, CA, USA

Na Meng Virginia Tech, Blacksburg, VA, USA

Bashar Nuseibeh The Open University, Milton Keynes, UK
Lero The Irish Software Research Centre, Limerick, Ireland

Leon J. Osterweil University of Massachusetts, Amherst, MA, USA

Doron A. Peled Bar Ilan University, Ramat Gan, Israel

José Miguel Rojas University of Leicester, Leicester, UK

Alireza Sadeghi University of California, Irvine, CA, USA

Anita Sarma Oregon State University, Corvallis, OR, USA

Nils Schwenzfeier Universität Duisburg-Essen, Essen, Germany

Tetsuo Tamai The University of Tokyo, Tokyo, Japan

Richard N. Taylor University of California, Irvine, CA, USA

Thein Than Tun The Open University, Milton Keynes, UK

Danny Weyns Katholieke Universiteit Leuven, Leuven, Belgium
Linnaeus University, Växjö, Sweden

Yijun Yu The Open University, Milton Keynes, UK

Tianyi Zhang University of California, Los Angeles, CA, USA

Process and Workflow

Leon J. Osterweil, Volker Gruhn, and Nils Schwenzfeier

Abstract Processes govern every aspect of software development and every aspect of application usage. Whether trivial, complex, formal, or ad hoc, processes are pervasive in software engineering. This chapter summarizes a variety of ways in which process models, also referred to as workflows, can be used to achieve significant improvements in a range of different disciplines. The chapter starts with a brief summary of the evolution of this approach over the past century. It then identifies some principal ways in which important advantages can be obtained from exploiting process models and process technology. The chapter goes on to specify key criteria for evaluating process modeling approaches, suggesting the different kinds of modeling approaches that seem particularly effective for supporting different uses. A representative set of examples of current process modeling approaches, and different ways in which process modeling is being used to good advantage, is then described. These examples are then used to suggest some key research challenges that need to be met in order for society to obtain a large range of further advantages from the continued development of process and process modeling technology.

1 Background, Goals, and Motivation

A nearly unique characteristic of humans is our ability to get things done by planning, coordinating, adapting, and improving how to achieve our objectives. It seems to be true that lions, wolves, and dolphins plan, coordinate, and execute schemes for killing prey to satisfy their needs for food, and that beavers make plans to build dams that raise water levels to expedite their access to trees needed

L. J. Osterweil
University of Massachusetts, Amherst, MA, USA
e-mail: ljo@cs.umass.edu

V. Gruhn · N. Schwenzfeier
Universität Duisburg-Essen, Essen, Germany
e-mail: gruhn@adesso-gmbh.de; Nils.Schwenzfeier@paluno.uni-due.de

S. Cha et al. (eds.), *Handbook of Software Engineering*,
https://doi.org/10.1007/978-3-030-00262-6_1

for food and shelter. But humans have carried things to a far higher level, using planning, coordination, and improvement as the basis for getting done nearly everything that we need or desire, ranging from the acquisition of food, to the construction of shelter, to the creation of entertainment vehicles, and even the creation of family and social units. Moreover, humans employ written media to supplement sight and sound as vehicles for improving coordination and expediting improvement. **In this chapter, we refer to this systematic and orderly approach to planning, coordination, and execution for the purpose of the improvement and effectiveness of functioning as *process*.** We note that the process approach has been exploited by governmental units of all kinds, by medical practice on all scales, by companies, large and small, (where the term *workflow* has been attached to this approach), and by the software development industry (where the term *software process* has been attached to this enterprise)—to enumerate just a few example communities.

1.1 Goals and Benefits of Process

The rather vague words "improvement" and "effectiveness" have just been used to characterize the goals of the diverse communities that have studied and exploited process. But we can be much more specific about the precise goals for the use of process to achieve improvement and effectiveness. We enumerate the most salient of these goals as follows.

1.1.1 Communication

A primary motivation for creating processes has been to use specifications of them as vehicles for improving communication among all process stakeholders (Indulska et al. 2009). The stakeholders for a process may be numerous and varied, including, for example, the current participants, proposed participants, parties intended to be the recipients of the results of the process, as well as auditors, regulators, and an interested public at large. The benefits to these stakeholders are also varied. Thus, for example, process participants can use a process specification to make it clear just what they are supposed to do when they are supposed to do it, and how they are supposed to do it. Recipients may wish to see the process specification in order to improve their confidence that they are being treated fairly and correctly. Regulators may wish to see the process specification to be sure that it conforms to applicable laws. It should be noted that different stakeholder groups will have different levels of technical sophistication, and different amounts of patience with details, suggesting that different kinds of process specifications are likely to be relatively more useful and acceptable to different stakeholder groups.

1.1.2 Coordination

Process specifications are also particularly useful in helping the members of a team to coordinate their efforts. Especially as the size of a team gets large, and the complexity of its task grows, it becomes increasingly important for all team members to understand what they are supposed to be doing, how their tasks relate to the tasks performed by others, how they are to communicate with each other, what can be done in parallel, and what must not be done in parallel. A good process specification can make all of that clear, greatly increasing the effectiveness of the team by reducing or avoiding duplication, delay, and error. As in the case of the use of process specifications to support communication, using them for coordination requires that the process specification be expressed in a form that is readily comprehensible to all process participants. This becomes increasingly difficult as the membership of a team becomes more diverse and distributed.

1.1.3 Training

Closely related to the previous two uses of process specifications for communication and coordination is the use of process specifications for training. In this case it is assumed that a process participant who is currently unfamiliar with, or insufficiently facile with, a process is able to use the process specification to gain a desired or needed level of understanding of how to participate effectively in the performance of the process (Jaakkola et al. 1994). Often, initial familiarization with the process is best communicated through a high-level, perhaps relatively informal, overall process specification. But when a sure grasp of the details of a process is needed, more precise, detailed, fine-grained process specifications are called for.

1.1.4 Understanding

Specifications of processes, especially particularly complex processes, can also be useful in supporting deeper understandings of the nature of a process. While an individual process participant may be able to readily understand his or her role in a process, and how to communicate effectively with others, this is no guarantee that any of the participants will have a deeper understanding of the overall nature of the entire process. A clear specification of the whole process can be used by participants to see the larger picture, and can be used by others, such as managers and high-level executives, to get a clear sense of its characteristics, its strengths, and its weaknesses. This, in a sense, is communication, but of a higher-level, more global sort of communication. Thus, process specifications that are able to present a clear high-level picture, while also supporting deep dives down into fine-grained details when desired, are relatively more effective in supporting understanding. Then too, since understanding is likely to be desired by both participants in, and observers of, the process, the form of the specification ideally ought to be something that is broadly understandable.

1.1.5 Improvement

The logical extension of the need to understand a process is the need to identify weaknesses and support process improvement through the remediation of identified weaknesses. Thus, improvement rests first upon analyzability, namely, the ability to examine a process specification sufficiently deeply and rigorously to support the identification of flaws, vulnerabilities, and inefficiencies, and then upon the ability to support making modifications to the process and verifying that the modifications have indeed removed the flaws, vulnerabilities, and inefficiencies (while also not injecting new ones). There are many dimensions in which process improvements are desired. Most fundamental is improvement through the detection and removal of defects that cause the process to fail to deliver correct results some of the time or, indeed, all of the time. Improvements to the speed and efficiency of a process are also frequent goals of process analysis and improvement. But increasingly there is a great deal of interest in detecting and removing process characteristics that render the process vulnerable to attacks, such as those that compromise organizational security and personal privacy.

Analyses of process specification can range from informal human inspections of informal process specifications all the way up through rigorous mathematical reasoning and verification of process specifications that are expressed using a rigorously defined formal notation. The former seems best supported by clear visualizations of the process, while the latter seems best supported by the use of rigorous mathematics to define both the process specification and specifications of desired properties. As both kinds of analysis seem highly desirable, a process specification that supports both clear visualization and rigorous mathematical analysis is also highly desirable. Such specifications are rare, however.

1.1.6 Guidance and Control

Process specification can also be used to help guide and control the performance of processes. In this way this use of process specifications differs fundamentally from the previously described uses. The previously described uses are offline, entailing only reading, analyzing, thinking about, and modifying a static descriptive entity. Process guidance and control, on the other hand, entails the dynamic use of a process specification to assume such roles as the coordination of the actions of participants, the evaluation of their performance, and the smooth integration of the contributions of both automated devices and humans. The greater the extent to which a process specification is relied upon for guidance and control, the greater must be the assurance that the specification is complete and correct. Thus, this use of a process specification typically relies upon careful and exhaustive analysis and improvement of the process specification. That, in turn, suggests that the process specification be stated in a particularly rigorous form, and be complete down to the lowest-level fine-grained details.

In this chapter, we will explore the approaches that have been taken in attempts to reach these specific goals, and will conclude the chapter with an exploration of where the attempts have fallen short and need to be improved and expanded. But first, some history.

2 History and Seminal Work

Work in the area of software process treats completed software systems as products that emerge as the result of the performance of a process. Work in the area of workflow treats the effective performance of business and management activities as the results of the performance of a workflow. In this way, software process and workflow are the rather direct extensions of much earlier work on other kinds of processes. It is hard to identify the earliest beginnings of this process approach, but for specificity in this chapter, we mark that beginning with the work of Frederick Taylor (1911). In this section of the chapter, we trace the development of process activity from Taylor to the present day. Figure 1 provides a visual summary of that history, which we will refer to throughout the rest of this chapter. The arrows in this figure can be thought of as representing how concepts and products have resulted from others. Specifically, when a concept or product is at the head of an arrow, it signifies that it has come into existence or practice at least in part due to the influence

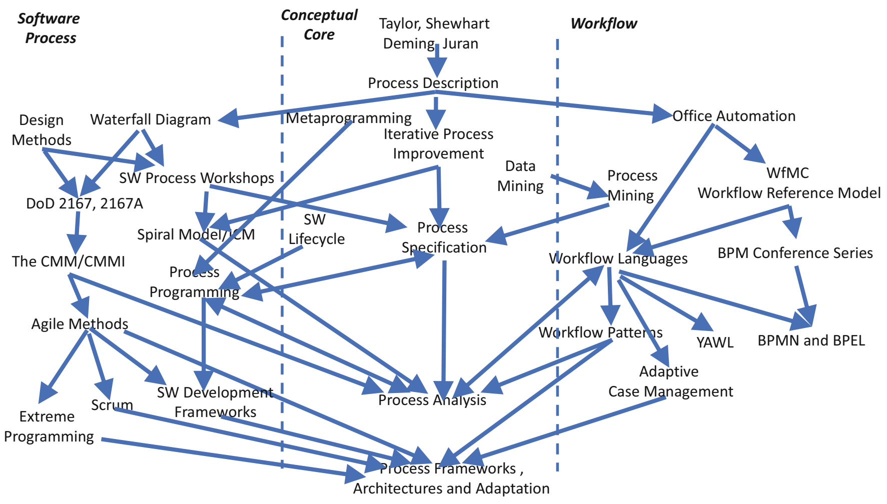

Fig. 1 Diagrammatic representation of the development of process thought and technology

of the concept or practice at the tail of the arrow. In some cases, double-headed arrows are used to indicate where concepts or practices have coevolved through strong and ongoing interactions. Note that the diagram is divided into three vertical columns, representing parallel development. The middle column represents the development of process core concepts, such as process definition, process analysis, and process evolution. The outer columns represent the parallel development of software process ideas and technologies (shown on the left) and workflow ideas and technologies (shown on the right). The figure indicates how core concepts have both emerged through consideration of practical technology development, and have also been the inspirations and stimuli for further development of technologies. The figure also indicates the almost complete lack of interactions between the workflow and software process communities. More will be said about this unfortunate situation in the rest of this chapter.

In the early twentieth century, Frederick Taylor looked at ways to understand and improve industrial processes (Taylor 1911). His work focused on measuring and improving productivity by scrutinizing the performance of both humans and machines, looking at ways to improve each and to use process improvement approaches to effect better coordination. Walter Shewhart (1931), W. Edwards Deming (1982), and Joseph Juran (1951) all also sought to improve processes, emphasizing how to use them to improve product quality, as well as process efficiency. The scope of their work encompassed industrial, management, and office processes, among others. Indeed this early work can be viewed as the forerunner of much later work on office automation, workflow, and business process management. Shewhart proposed the Shewhart Cycle, which is still referred to as a clear model of continuous improvement. It posits that process improvement is a four-phase process. The first phase of the process entails examination and analysis of the process and its behavior with the aim of identifying shortcomings that are in need of being remedied. The second phase entails the proposal of specific remedies for the identified shortcomings. In the third phase, the proposed remedies are tried and evaluated to determine whether they effect the desired improvements. Only if the desired improvement has been demonstrated can the fourth phase be initiated. The fourth phase entails the incorporation of the remedies into the current process, resulting in a new process. This new process then becomes the subject of the first phase of a new improvement cycle. Process improvement, thus, is expected to be an ongoing process, continuing for the lifetime of the process. The Shewhart Cycle has been reinvented with various minor modifications and renamings continuously over the past 100 years.

With the work of Deming and Juran we see the beginnings of the use of notation and crude formalisms to attempt to describe processes more clearly and precisely, in order to improve the effectiveness of efforts to use them for better communication, coordination, education, and improvement. As noted in Fig. 1, this work is one of the earliest examples of the use of visual notation to represent processes. Deming and Juran had a powerful influence on manufacturing through the first half of the twentieth century. In particular, their argument that process improvement could be used to improve product quality was eagerly taken up by manufacturers in Japan during the mid-twentieth century. The marketplace success of such Japanese products as cars, cameras, and electronics was widely attributed to the quality of these products, which was in turn widely attributed, at least in large measure, to a focus on process improvement. This focus on process improvement, consequently, attracted the attention of American and other manufacturers, and led to a wider appreciation of the ideas of Juan and Deming, and to a growing focus on the importance of process. All of this has set some important directions that continue through today.

The birth of the software industry in the mid-twentieth century came against this backdrop of a strong focus on quality and the use of systematic process improvement to achieve that quality. So, it is not hard to understand that there was an early focus on processes aimed at assuring software development efficiency, and software product quality. An early representative of this focus was the enunciation of the Waterfall Model (Royce 1970, 1987) of software development, which as noted in Fig. 1, used a visual notation to represent processes. In its earliest and most primitive form, the Waterfall Model simply mandated that development must begin with requirements specification, then proceed through design, then on to coding, and finally to deployment and maintenance. It is worthwhile to note that this kind of high-level process had previously been adopted by the US Department of Defense as a guide to the creation of hardware systems. Thus, its enunciation as a model for software development is not hard to understand. Almost immediately, however, the manifest differences between hardware items and software systems occasioned the need for a more careful examination of what a software process specification should look like. As a result, successive modifications to the basic Waterfall Model added in such modifications and embellishments as iterations of various kinds, decomposition of the major phases into subphases, and identification of types of artifact flows through the phases and subphases.

Much work on embellishing the Waterfall Model during the 1970s and 1980s focused on systematizing and formalizing the software design process. Three examples of this work are the SADT approach (Ross and Schoman 1977), proposed by Douglas Ross; the JSP system (Jackson 1975), proposed by Michael Jackson; and the Modular Decomposition method (Parnas 1972), proposed by David Parnas. Each suggested canonical sequences of artifacts to be created, with the end result expected to be superior designs. It is particularly interesting that the main focus of these efforts was on the artifacts to be created, rather than on the activities to be carried out. Each approach focused attention on the structure, semantics, and contents of the intermediate and final artifacts to be created during design. In contrast to most other early process specification work, there is correspondingly little attention paid to the structure and detail of the sequence of activities to be performed, which are addressed essentially as sparingly specified transformers of the input artifacts they receive into the outputs that they create. It is noteworthy, in addition, that the SADT and JSD methods incorporated pictorial diagrams of their artifacts.

Much of this embellishment and amplification was incorporated into a succession of US Department of Defense standards, such as DoD Standard 2167 and 2167A (DoD 2167, DoD 2167A). The totality of these embellishments resulted in documents with so much complexity and flexibility that they barely served as useful guides to how software development should actually proceed.

DeRemer and Kron (1976) made an important conceptual step forward with their identification and articulation of “Metaprogramming,” which they described, essentially, as the process by which software is produced. In their paper they suggest that the process by which software is produced seems to be an entity whose nature might be better understood by thinking of it as some kind of a program, and the creation of this process as some kind of programming.

A sharp focus on the concept of a *Software Process* began in the 1980s. One particularly powerful catalyst was the establishment of the Software Process Workshop series, which was first held in 1984, organized by Prof. Meir (Manny) Lehman of Imperial College London (ISPW 1). Interest in this area was intensified by Prof. Leon Osterweil’s keynote talk at the 9th International Conference on Software Engineering, which suggested that “Software Processes are Software Too” (Osterweil 1987), elaborating on the suggestion made by DeRemer and Kron. As indicated in Fig. 1, this idea of *Process Programming* drew upon basic notions of a software development life cycle, and led to various conjectures about the nature of languages that might be used, not simply to describe, but to *define* software processes, a search for a canonical, “ideal” software process, and initial investigations of how to analyze, improve, and evolve software processes.

Interestingly at about this same time, work began at Xerox Palo Alto Research Center on *Office Automation.* Dr. Clarence Ellis and his colleagues began pondering the possibility of creating a notation that could be used to specify organizational paperwork processing sufficiently precisely that the work could be shared smoothly and more efficiently among humans and machines (especially Xerox copiers!) (Ellis and Nutt 1980; Ellis and Gibbs 1989; Ellis et al. 1991). That line of work continued in parallel with similar work on Software Process through the 1980s and into the 1990s. Much of the work was organized and led by the Workflow Management Coalition, which enunciated a canonical business process workflow architecture (Hollingsworth 1995). In the 1990s, there was a brief and unsuccessful attempt to integrate the Workflow and Software Process communities. Office Automation was subsequently renamed Business Process Management, and has grown into a large and powerful community with its own meeting and publication venues (BPM, Conferences, Newsletters), almost completely separate from the Software Process community’s meeting and publication venues (ISPW, ICSSP, JSEP). Unfortunately, the Business Process and Software Process communities continue to have few interactions, sharing very little in the way of work, ideas, and practitioners, despite the fact that their problems, and even many of their approaches, are very much the same.

The diagram shown in Fig. 1, summarizing much of the above history, will also be referred to occasionally to establish context for many of the key concepts and achievements to be described below. To facilitate being precise and clear about all of this, we now state some definitions and establish some terminology.

3 Some Definitions, Unifying Assumptions, and Characterizations

In order to improve the precision and specificity of the material to be presented in the rest of this chapter, it is important to establish some terms and concepts.

3.1 Processes and Workflows

Because both the entity that is the focus of the Software Process community and the entity most commonly referred to as a "workflow" by the Business Process Management community seem to be nearly identical, we will use the word "process" to refer to both of them for the rest of this chapter. As shall be explained subsequently, some distinctions can be drawn between the views of the two communities about these entities. In such cases, this chapter will identify these differences and distinctions. Otherwise, our use of the word process will be intended to refer to both a software process and a business workflow.

3.2 Process Performances

It is important from the outset to distinguish carefully between a process, a performance of a process, and a specification of a process. A *process* can be thought of as a collection of activities taking place in the real world, entailing the coordination and sequencing of various tasks and activities that may involve work by both humans and automated systems, whose aim is to produce one or more products or to achieve some desired change in the state of the real world. We note that a process may be performed somewhat differently under differing circumstances (e.g., the abundance or scarcity of task performers, the existence of different real-world conditions, or the occurrence of unusual events), and we refer to such a single, particular, specific performance, yielding a particular specific outcome, as a *process performance*.

3.3 Process Specifications

A *process specification* is an abstraction, intended to represent all possible process performances. Note that a process may allow for a great variety (perhaps even an infinite number) of different process performances, depending upon the particular combination of performers, contexts, and event sequences present at the time of a specific process performance. A process may allow for so many different performances that it is quite possible that a process will include some performances that have not yet ever been experienced at the time that a specification of the process is created. Indeed, the need to anticipate performances that have not yet occurred, but might be problematic, is often the reason for creating a process specification. A process specification is typically stated either informally, in natural language text, or by means of an abstract notation that may or may not have rigorous semantics. Moreover, the specification may or may not have a visual representation.

3.4 Activities

Typically, a process specification aims to integrate specifications of activities, artifacts, and agents, and the relations among them. Activities are the specific tasks, steps, or actions whose completion must occur in order for the process to be performed. Thus, a software process will typically contain such steps as "Produce code" and "Execute test cases," while a business process will typically contain steps such as "Approve payment," and "Create invoice." Typically, a key aspect of a process specification is the way in which these activities are arranged into structures. Most commonly, a process's activities are structured using flow-of-control constructs such as sequencing and alternation, while the use of concurrency and iteration constructs is not uncommon. In addition, many process specifications also use hierarchical decomposition to support the specification of how higher-level activities can be comprised of aggregations of more fine-grained, lower-level activities. This supports the understanding of higher-level notions by drilling down into successively lower-levels of detail. Thus, the software process step "Execute test cases," will have such substeps as "Execute one test case" and "Create summary results," and the business process "Send invoice" will have such substeps as "Create itemized list" and "Acquire authorization to send."

3.5 Process Artifacts

Most process specifications also specify how artifacts are generated and consumed by the process's activities. This is particularly important for processes whose goals include the creation of such artifacts. Thus, a software process will incorporate such artifacts as code, architecture specifications, and documentation. A business process will incorporate such artifacts as invoices, approvals, and status reports. Typically, the artifacts are represented only as inputs and outputs to and from the different activities of the process. Indeed, it is often the case that the semantics of an activity are defined implicitly to be the transformation of its inputs into its outputs. Similarly, the semantics of an artifact are often not defined more formally than as an entity that is generated and used as specified by activity inputs and outputs. On the other hand, there are some processes such as some early software design methods (e.g., SADT and JSD, addressed above) that emphasize careful specification of artifacts, rather than of activities. In such cases, the semantics of these artifacts may be elaborated upon, typically by hierarchical decompositions of higher-level artifacts into their lower-level artifacts and components, but also often by prose explanations and examples. In business processes, such hierarchical specifications may be further augmented by database schema definitions.

3.6 Process Agents

A process specification's agents are the entities required in order for the activities to be performed. Not all process specifications require the specification of an agent for every activity. But a typical process will incorporate some steps for which the characteristics of the entity (or an explicit designation of the exact entity) needed to perform the step must be specified explicitly. Thus, for example, in a software process the "Authorize release of new version" step may require a high-level executive as its agent, and the "Execute one test case" step may require a computing device having some stated operational characteristics. Similarly, in a business process, the "Release paychecks" step may require a high-level financial executive as its agent, and the "Affix postage" step may require an automatic postage machine as its agent. It should be noted that the performance of some steps may require more than one agent. Thus, for example, "Create design element" will require not only a human designer, but may also require the use of an automated design aid. Some process specifications support the specification of such auxiliary resources as well.

The next section presents a framework incorporating the principal concepts underlying work in the process domain. It explores these concepts, indicates how they relate to each other and how they evolved. The contributions of some of the principal projects in each area are presented.

4 Conceptual Framework

Referring again to the middle column of Fig. 1, we see that technology in the software process and workflow areas have both benefitted from and nourished the development of key process concepts such as process specification, process analysis, and process frameworks. This section presents these key concepts, indicating some ways in which they have coevolved with technology development.

4.1 Process Specification Approaches

As noted in the preceding section, different approaches to specifying processes seem to be more appropriate to some uses than to others. In this section, we summarize some of these approaches, their strengths and weaknesses, and suggest the uses to which they seem most suitable.

4.1.1 Process Specification Evaluation Criteria

It is important to emphasize that a process specification is an abstraction whose goal is to approximate the full range of characteristics and properties of an actual process. Different forms of specification support this approximation to different extents. It has previously been suggested (Osterweil 2009) that four principal dimensions should be used to categorize the extent to which a process specification approach approximates process specificity:

- *Level of detail:* Most process specifications support hierarchical decomposition, at least of process steps. More detailed and specific process specifications incorporate more hierarchical decomposition levels. Many of the goals for process work seem to be met most surely and easily if the process is specified down to a fine-grained level.
- *Breadth of semantics:* While most process specifications integrate specification of activities and artifacts, some also incorporate agent and resource specifications. Some go further still, incorporating such additional semantic issues as timing, physical locations, etc.
- *Semantic precision:* The understanding of some process specifications relies upon human intuition, leaving relatively more room for different interpretations. Other specifications use notations such as Finite State Machines (FSMs) and Petri Nets whose semantics are based upon rigorous mathematics, enabling the unambiguous inference of definitive interpretations.
- *Comprehensibility:* Most process specifications incorporate some kind of visualization, while some (especially those based upon rigorous mathematical semantics) lack such visualizations, typically using mathematical notations instead. The existence of a visualization is particularly important in cases where domain experts, generally not well-trained in mathematics, are needed in order to validate the correctness of the process specification.

4.1.2 Example Process Specification Approaches

We now present a representative sample of different approaches that have been taken to the specification of processes, and use the preceding four dimensions to characterize and classify them. As shown in Fig. 1, specific approaches taken in both the software development and business process communities have contributed to our overall understanding of the key issues in process specification. Conversely, these understandings have helped to sharpen the approaches.

Natural Language

The most straightforward and familiar form of process specification is natural language (e.g., English). Indeed, natural language specifications of processes abound and are applied to specifying processes in both software development and business process domains, as well as in myriad other domains. The key strength of this specification approach is breadth of scope, as natural language can be used to describe pretty much anything. While it might appear that comprehensibility and depth of details are other key strengths of natural language, it is important to acknowledge that they are undercut by a key weakness, namely, the lack of precise semantics for natural languages. This weakness leaves natural language specifications open to different interpretations, thus making the comprehensibility of such specifications illusory, often leading to misinterpretations and disagreements, and often making attempts to clarify low-level details frustrating exercises in trying to be very precise while using a descriptive medium that is incapable of the needed precision. Hence, it is increasingly common for process specification users to augment or replace natural language with some forms of notation that are intended to address the lack of precise semantics in natural language.

Box-and-Arrow Charts

Rudimentary diagrams have been used widely to specify processes from many different domains. The most basic of these diagrams represents a process activity as a geometric figure (typically a rectangle), and represents the flow of artifacts between activities by arrows annotated with identifications of these artifacts. They key advantages of this diagrammatic notation are its relative comprehensibility, and the existence of some semantics that can be used as a basis for attaching some unambiguous meaning to a diagram. It should be noted, however, that it is important for these semantics to be specified clearly, as box-and-arrow diagrams are often used to represent control flow, but are also not uncommonly used to represent data flow. Breadth of semantics and depth of detail are greatly increased when these diagrams comprise more than one kind of geometric figure and more than one kind of arrow. Thus, for example, representing alternation in process control flow is more clearly represented by a choice activity, often represented by a diamond-shaped figure. Similarly, concurrency is more clearly represented by special edges (e.g., those that represent forking and joining concurrent activities). Decomposition of both activities and artifacts down to lower levels of detail is more clearly and precisely specified by still other kinds of edges that represent this additional kind of semantics. Indeed, one can find a plethora of different box-and-arrow approaches, featuring different choices of kinds of boxes, kinds of arrows, and specific semantics for specifying the meanings of all of them.

Finite State Machines

Finite state machines (FSMs) have also often been used to specify processes. This approach is particularly effective in cases where the performance of a process is comfortably and naturally thought of as the effecting of changes in the state of the world. In this view, a world state is thought of as a structure of values of artifacts and parameters, and it is represented by a small circle. If the performance of a single activity of the process is able to move the world from one state directly to another, then the circles representing these two states are connected by an arrow. A distinct advantage of this form of process specification is its rigor, derived from the existence of a precise semantics, and the consequent existence of a large body of mathematical results that can then be used to support the inference of some precise characteristics of the process. Unfortunately, it is not uncommon for process practitioners from domains other than Computer Science to misunderstand the semantics of these diagrammatic icons, confusing states for activities, and state transitions for data flows. Given these misunderstandings, unfortunately the understandability of FSMs is far less universal that might be desired. Depth of detail is also facilitated by the use of hierarchical FSMs. But FSMs are inherently nonhierarchical, and thus additional semantics are needed to support process specifications that use the hierarchical decomposition of FSMs. Similarly, additional semantics are needed to support specifying artifact flow, timing, resource utilization, and many other dimensions of semantic breadth and depth. Here too, a large number of different enhancements of the basic FSM semantics have been proposed and used to support the specification of processes.

Petri Nets

Petri Nets (1962) have also been used widely to support the precise specification of processes (Murata 1989). Here too, a major advantage of doing so is the existence of both powerful semantics and a large body of mathematics, all of which can be used to support inferring precise understandings of processes specified by the Petri Net. Here too, however, these understandings are relatively inaccessible to experts from domains outside of computing. Moreover, a basic Petri Net has relatively restrictive semantics, not including, for example, hierarchical decomposition, specification of artifacts, and a number of other kinds of semantics needed to support the breadth and depth that is often desired in a process specification. For this reason a variety of enhancements to a basic Petri Net have been proposed and evaluated (David et al. 2005). Thus, there are Colored Petri Nets that support a primitive system of artifact types, Hierarchical Petri Nets that support the specification of hierarchical decomposition, and still more elaborate enhancements. As might be expected, these more elaborate Petri Nets are more effective in supporting the specification of more process semantic issues to greater levels of detail.

Business Process Modeling Notation

Decades of experimentation with reusing existing notations (such as the previously described box-and-arrow charts, FSMs, and Petri Nets) to attempt to specify processes eventually caused some process practitioners to realize that the clear, precise, and complete specification of their processes would be expedited considerably by the creation of a special purpose process specification notations. Principal among these has been the Business Process Modeling Notation (OMG). BPMN features the use of intuitive graphical icons and notations to express a broad range of process semantics, including concurrency, hierarchical decomposition, artifact flow, and agent assignment to activities. As such it has constituted a large step forward in the effective specification of processes, accessible both to process practitioners and experts from noncomputing domains. A major failing of BPMN, however, is the lack of a formal definition of the language semantics. Thus, while BPMN process specifications offer broad intuitive appeal, they still suffer from the absence of a semantic basis that could be used to resolve any doubts, ambiguities, or disagreements about the precise meaning of a process specification. Still, the appealing visualizations afforded by BPMN specifications have helped to make BPMN a popular choice for use in describing complex processes.

Programming Languages

The foregoing progression of process specification approaches suggests a growing awareness that processes are multifaceted entities, requiring significant power and sophistication for their sufficiently complete specification, and significant formalization for their precise understanding. From this perspective it is not much of a leap to suggest, as did Osterweil in 1987, that programming languages might be used to specify processes (Osterweil 1987). In this section, we describe some representative efforts to do just that.

- **Business Process Execution Language (BPEL**): BPEL is a programming language specially designed for use in supporting the specification of processes, most specifically business processes (Andrews et al. 2003). BPEL has well-defined semantics that are indeed sufficiently strong to support the actual execution on a computer of a BPEL specification. To emphasize this important distinction, we will refer to such an executable process specification as a *process definition*. BPEL has powerful semantics that support such important, but challenging, process features as concurrency, exception handling, and the specification of resources. A major BPEL drawback is the lack of a visual representation of the defined processes. Lacking an appealing visualization, BPEL is of value mostly to process domain experts, and is relatively inaccessible to experts from most other domains. There has been some effort to merge the BPMN and BPEL technologies, essentially using BPEL to provide the semantics, and the executability, that BPMN lacks. To date, this effort has met with only limited success.

- **YAWL (Yet Another Workflow Language):** YAWL is anything but "yet another" workflow language (van der Aalst et al. 2005). It succeeds in providing a powerful, semantically strongly defined process definition language that provides strong support for such difficult process constructs as concurrency, exception management, and agent specification. In addition, however, it is also accompanied by a very appealing visual representation, and a suite of documentation and user manuals that make the language highly accessible. Its strength in all of these dimensions makes it almost unique among process specification approaches.
- **Little-JIL:** Little-JIL (Wise et al. 2000; Little-JIL 2006) is another specially designed language for supporting the development of executable process definitions. Like YAWL, it features a visual representation that is designed to make process definitions readily accessible by experts in domains other than computing. Taking the idea that processes should be thought of as first-class programmable entities, Little-JIL borrows heavily upon basic concepts that have proven to be particularly useful in traditional computer programming languages. Thus, for example, abstraction is a first-class concept, supported strongly and clearly in Little-JIL. Similarly, the language takes a powerful and flexible approach to the specification and handling of exceptions and concurrency. Particular attention is also paid to the specification of agents, both human and nonhuman, and the specification of precisely how they are used in the execution of a process. Unfortunately, Little-JIL is not well-supported by manuals, documentation, and other accompaniments generally expected of a language that might see widespread use.

4.2 Process Acquisition

Traditionally, understandings of processes have been acquired by some combination of observing the performers, interviewing the performers, and reading documentation, most often written in natural language. These approaches are complementary, with each having the potential to contribute important knowledge and insights, but each lacking important dimensions. Thus, from observation of actual performers it is possible obtain a strong sense of how a process proceeds, especially in the normative cases, where little or nothing unusual arises and needs to be dealt with. In cases where the process is particularly complex, and may require the collaboration of multiple performers, it is generally the case that written documentation is important, providing high-level concepts and structures that the process elicitor can use to help structure and organize the relatively low-level activities that are being observed.

Written documentation typically has significant limitations, however, for all of the reasons addressed in our immediately preceding discussions of the different approaches to process specification. Processes tend to be large and complex entities with a range of semantic features and issues that render them very hard to describe clearly and completely. Thus, written documentation, even documentation accompanied by visual representations, typically falls far short of the completeness and precision needed. The failure of most documentation to address such issues as exception handling is not surprising in view of the difficulty of doing so. As a consequence, process elicitation then also often entails interviewing process performers.

Interviewing process performers offers the opportunity to delve into process details to whatever level of depth may be desired, and also affords a chance to explore nonnormative exceptional situations, including those that may not ever have occurred previously, and thus would be impossible to observe. Given that a process may be highly variable, and need to deal with an almost limitless variety of nonnormative situations, it is inevitable that process elicitation must be regarded as an ongoing process of its own, uncovering a continuous stream of new information that will need to be incorporated into an ever-evolving process specification.

That realization has given rise to an important new direction in process elicitation, namely, process mining. Process mining is the acquisition of process specification information through the analysis of large quantities of process event data, accumulated over a long period of time from the actual performance of large numbers of process instances. The actual process specification information that is acquired depends upon the nature of the process event data that has been accumulated as well as the semantic features of the process specification approach used. Cook and Wolf (1995) seem to have been among the first to attempt to mine process specification features, attempting to capture a process specification in the finite state machine formalism. More recently, especially with the rise in the popularity and effectiveness of data mining and associated knowledge discovery technologies, the acquisition of process specification information has become a very widespread pursuit.

As shown in Fig. 1, process mining has been explored extensively in the BPM community where datasets, often quite massive, archiving perhaps years of experience with certain processes, have been mined, often recovering substantial process specification information. This approach seems particularly appropriate for recovering details about relatively straightforward processes with few alternations of control, and with relatively few nonnormative events that must be detected and responded to. In processes where external events cause the creation of nonnormative process execution states, the process responses might need to become quite large and complex, reflecting the different specialized responses that might be needed to deal with a plethora of different process states, created by different sequences of events. Some process responses to formalisms are better equipped than others to specify such processes clearly and concisely.

It is important, moreover, to recognize that the process data that is acquired through process mining can only express actual experiences with process executions that have occurred. Thus, this approach would not be very helpful in preparing process performers for events and scenarios that have not yet taken place. Process mining also cannot be expected to be particularly successful in identifying worrisome vulnerabilities to events and event sequences that have not already occurred. This suggests that process mining is probably not the preferred way to study process vulnerability and robustness.

4.3 Process Analysis Facilities and Results

Especially because of the previously discussed difficulties in identifying a superior process specification approach, and in using it to acquire a clear, complete, and detailed process specification , it is all the more important to also explore approaches to gaining assurances about the correctness of such process specifications. Indeed, as shown in Fig. 1, the enterprise of process analysis occupies a very central position in the process domain. Understanding of this enterprise is seen to be nourished by work in both the software process and workflow domains. Moreover, the development of process analysis technologies in both of these areas has been advanced by understandings of the nature of the enterprise of analysis. Thus, for example, in striking analogy to approaches to analysis in such domains as mechanical engineering and software engineering, the approaches to analysis of process specification can be neatly classified into dynamic and static approaches. Borrowing from software engineering, one can further categorize the static approaches into syntactic approaches, and various kinds of semantic approaches.

It should be noted that the creation of such analyzers is a nontrivial project. Thus, the prior existence of analyzers for such existing formalisms as Petri Nets and Finite State Machines seems to have improved the attractiveness and popularity of these approaches as vehicles for process specification.

4.3.1 Dynamic Analysis of Process Specifications

The principal approach to the dynamic analysis of process specifications is discrete event simulation. Such a discrete event process simulation must maintain the state of the process as each of its activities is performed. The state is usually characterized by the simulated values of all of the artifacts managed by the process, the estimated time that has elapsed through each process activity, and sometimes also by the states of the agents who are involved in performing the process.

As noted above, any given process might be performed differently due to different behaviors of different agents, the occurrence of different combinations of external events, and differences in the states of input artifacts at the commencement of execution. Thus, a process simulator must be equipped to support exploration of a variety of different execution scenarios. This is accomplished in a variety of ways. Most simulators allow for the specification of different statistical distributions