Factors Influencing Software Development Productivity - State of the Art and Industrial Experiences

Adam Trendowicz, Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering

Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

+49-631-6800-0

{adam.trendowicz; juergen.muench}@iese.fraunhofer.de

ABSTRACT

Managing software development productivity is a key issue in software organizations. Business demands for shorter time-to-market while maintaining high product quality force software organizations to look for new strategies to increase development productivity.

Traditional, simple delivery rates employed to control hardware production processes have turned out not to work when simply transferred to the software domain. The productivity of software production processes may vary across development contexts dependent on numerous influencing factors. Effective productivity management requires considering these factors. Yet, there are thousands of possible factors and considering all of them would make no sense from the economical point of view. Therefore, productivity modeling should focus on a limited number of factors with the most significant impact on productivity.

In this chapter, we present a comprehensive overview of productivity factors recently considered by software practitioners. The study results are based on the review of 126 publications as well as international experiences of the Fraunhofer Institute, including the most recent 13 industrial projects, 4 workshops, and 8 surveys on software productivity. The aggregated results show that the productivity of software development processes still depends significantly on the capabilities of developers as well as on the tools and methods they use.

Keywords

Software, development productivity, influencing factors.

TABLE OF CONTENT

Author's version of an article published in Advances in Computers, (Marvin V. Zelkowitz, ed.), Elsevier, vol. 77, pp. 185-241, 2009, DOI: 10.1016/S0065-2458(09)01206-6

http://www.sciencedirect.com/science/article/pii/S0065245809012066

1. INTRODUCTION

Rapid growth in the demand for high-quality software and increased investment in software projects show that software development is one of the key markets worldwide [1], [54]. Together with the increased distribution of software, its variety and complexity are growing constantly. A fast changing market demands software products with ever more functionality, higher reliability, and higher performance. Software project teams must strive to achieve these objectives by exploiting the impressive advances in processes, development methods, and tools. Moreover, in order to stay competitive and gain customer satisfaction, software providers must ensure that software products with a certain functionality are delivered on time, within budget, and to an agreed level of quality, or even with reduced development costs and time. This illustrates the necessity for reliable methods to manage software development productivity, which has traditionally been the basis of successful software management. Numerous companies already measure software productivity [102] or plan to measure it for the purpose of improving their process efficiency, reducing costs, improving estimation accuracy, or making decisions about outsourcing their development.

Traditionally, the productivity of industrial production processes has been measured as the ratio of units of output divided by units of input [95]. This perspective was transferred into the software development context and is usually defined as *productivity* [67] or *efficiency* [123]. As observed during an international survey performed in 2006 by the Fraunhofer Institute for Experimental Software Engineering, 80% of software organizations adapts this industrial perspective on productivity to the context of software development, where inputs consist of the effort expended to produce software deliverables (outputs). The assumption those organizations make is that measuring software productivity is similar to measuring any other forms of productivity. Yet, software production processes seem to be significantly more difficult than production processes in other industries [1][8][28]. This is mainly because software organizations typically develop new products as opposed to fabricating the same product over and over again. Moreover, software development is a human-based ("soft") activity with extreme uncertainties from the outset. This leads to many difficulties in the reliable definition of software productivity. Some of the most critical practical consequences are that software development productivity measures based simply on size and effort are hardly comparable [8], and that size-based software estimates are not adequate [90].

These difficulties are related to a variety of practical issues. One of these is software sizing[1]. Large numbers of associated, mutually interacting, and usually unknown factors influencing software productivity

[1] Software sizing (size measurement) as a separate large topic is beyond the scope of this chapter. For more details on the issue of size measurement in the context of productivity measurement, see, for example [73].

(besides size) are another critical issue. Even if reliable and consistent size measurement is in place, the key question of software productivity measurement usually remains and may be formulated as follows: "What makes software development productivity vary across different contexts?" In other words, which characteristics of the project environment capture the reasons why projects consumed different amounts of resources when normalized for size.

To answer this question, characteristics that actually differentiate software projects and their impact on development productivity have to be considered. Those characteristics include personnel involved in the project, products developed, as well as processes, technologies and methods applied.

Yet, there are several pragmatic problems to be considered when analyzing software project characteristics and their impact on productivity. One is, as already mentioned, the practically infinite quantity of potential factors influencing software productivity [98]. Even though it is possible to select a limited subset of the most significant factors [65][71], both factors and their impact on productivity may differ depending on productivity measurement level and context [8]. This also means that different project characteristics may have different impacts on the levels of single developer, team, project, and whole organization. Moreover, even if considered on the same level, various productivity factors may play different roles in the embedded software, in web applications, and in waterfall or incremental development. This implies, for example, that factors and their ratings covered by the COCOMO model [23] would require reappraisal when applied in a context other than that in which the model was built [80].

To address this issue, considerable research has been directed at identifying factors that have a significant impact on software development productivity. This includes (1) studies dealing directly with identifying productivity factors [72], (2) studies aiming at building cost and productivity modeling and measurement techniques, methods, and tools [28], (3) data collection projects intending to build software project data repositories useful for benchmarking productivity and predicting cost [48][56][68][118], as well as (4) studies providing influence factors hidden in best practices to reduce development cost (increase productivity) [71].

Although a large amount of work has been directed recently at identifying significant software productivity factors, many software organizations actually still use the simplified definition of productivity provided in [65]. In the context of high-maturity organizations [102], where production processes are stable and projects largely homogeneous, a simple productivity measure may work, provided that it is used to compare similar projects. Yet, in the case of cross-context measurement of heterogeneous projects with unstable processes, using simple productivity may lead to serious problems. In consequence, organizations that failed to measure simplified productivity find it either difficult to interpret and/or entailing little benefit [8].

Software organizations do not consider productivity factors because they often do not know which ones they should start with – which are the most significant ones. Existing publications provide hardly any support in this matter. Published results are distributed over hundreds of publications, which in many cases lack a systematic approach to presenting influence factors (e.g., context information, relative impact on productivity) and/or present significant factors implicitly as cost drivers in a cost model, project attributes in a data repository, or best practices for improving development processes.

In this study, we try to provide a systematic overview of those productivity factors that have the most significant impact on software productivity. In doing so, we assume that software cost is a derivative of software productivity and, therefore, we do not treat factors influencing cost (also called cost drivers) separately. Besides commonly available published studies investigating productivity factors, the overview also includes internal experiences gained in recent years at the Fraunhofer Institute for Experi-

mental Software Engineering (IESE) in several industrial projects, workshops, and surveys performed in the area of software productivity measurement and cost modeling. Yet, due to confidentiality reasons, we do not disclose some sensitive data such as company name or titles of respective internal reports that may indicate the names of the involved industry partners. Instead, we refer to certain characteristics of the companies. Such context information might be useful when selecting relevant factors for similar contexts, based on the overview presented in this chapter.

The remainder of the chapter is organized as follows. Section 2 presents the design of the study. Section 3 provides brief definitions of the terminology used. Section 4 and Section 5 provide a summary of productivity factors based on the literature review and the authors' industrial experiences gained at Fraunhofer IESE, respectively. An indepth discussion on selected productivity factors presented in the literature is given in Section 6. Section 7 provides an overview of several practical issues to be considered when identifying and productivity factors and using them to model software development productivity and cost. Finally, Section 8 summarizes the presented results and discusses open issues.

2. DESIGN OF THE STUDY

The review of productivity factors presented in this chapter consists of two parts. First, we present a review of the authors' individual experiences gained during a number of industrial initiatives. Here we would include factors identified for the purpose of cost and productivity modeling as well as factors acquired during surveys and workshops performed by Fraunhofer IESE in the years 1998-2006 (also referred to as *IESE studies*). The second part presents an overview of publications regarding software project characteristics that have an influence on the cost and productivity of software development.

2.1 Review of Industrial Experiences

The review includes the following industrial initiatives:

- International commercial projects regarding cost and/or productivity modeling performed in the years 1998-2006 at medium and large software organizations in Europe (mainly Germany, e.g., [27]), Japan, India, and Australia (e.g., [110]).
- International workshops on cost and/or productivity modeling performed in the years 2005-2006 in Germany and Japan (e.g., [6][8][66]). The workshop results include factors explicitly encountered by representative of various, international software organizations. The considered factors consist of both factors that were considered by experts as having a significant impact on productivity but not already measured, and factors already included in the organizational measurement system for managing development cost and productivity.
- Surveys on productivity measurement practices performed in various international companies in the years 2004-2006. These surveys concerned productivity/cost modeling and included questions about the most significant factors influencing software productivity. The 2004 study was a state-of-the-practice survey performed across 12 business units of a global software organization dealing mainly with embedded software. The four 2005 studies were performed across Japanese units of internationally-operating software organizations dealing both with embedded and business applications. Finally, the 2006 survey was performed across 25 software organizations all over the world and covered various software domains.

2.2 Review of Related Literature

2.2.1 Review Scope and Criteria

The design of the review is based on the guidelines for structural reviews in software engineering proposed by Kitchenham [78]. Yet, based on the conclusions of Jørgensen and Shepperd [73], we relied on a manual rather than an automatic search of relevant references. The automatic search through the INSPEC repository was complemented by a manual search through references found in reviewed papers and using a generic web search machine (www.google.com). The automatic search criteria were specified as follows:

INSPEC**: (SOFTWARE AND ((COST OR EFFORT OR PRODUCTIVITY) WITH (FACTORS OR INDICATORS OR DRIVERS OR MEASURE))).TX.* ***= 417 documents

The available papers[2] were selected for inclusion (full review) based on the title and abstract. The review was done by one researcher. The criteria used to decide whether to include or exclude papers were as follows:

- **Availability and Age**: The papers had to be published no earlier than 1995. Since the type and impact of influence factors may change over the time [86], we limit our review to the past decade.

  We made an exception for papers presenting software project data sets. Although indirectly, we did include those that were collected and first published before 1995 but that were applied to validate cost/productivity models published after 1995.

- **Relevancy**: The papers had to report factors with a potential impact on software development cost and/or productivity. Implicit and explicit factors were considered. *Explicit* factors are those considered in the context of productivity modeling/measurement. *Implicit* factors include those already included in public-domain cost models and software project data repositories. *Algorithmic cost estimation models*, for example, include so-called cost drivers to adjust the gross effort estimated based only of software size. *Analogy-based methods*, on the other hand, use various project characteristics found in the distance measure, which is used to find the best analogues to base the final estimate on. Common software project data repositories indirectly suggest a certain set of attributes that should be measured to assure quality estimates.

- **Novelty**: We did not consider studies that adopt a complete set of factors from an other reference and do not provide any novel findings (e.g., transparent model) on a factor's impact on productivity. We do not, for instance, consider models (e.g., those based on neural networks) that use as their input the whole set of COCOMO I factors and do not provide any insight into the relative impact of each factor on software cost.

- **Redundancy**: We did not consider publications presenting the results of the same study (usually presented by the same authors). In such cases, only one publication was included in the review.

- **Perspective**: As already mentioned in the introduction, there are several possible perspectives on productivity in the context of software development. In this review we will focus on project productivity, i.e., factors that make development productivity differ across various projects. Therefore, we will, for example, not consider the productivity of individual development pro-

[2] Unfortunately, some publications could not be obtained though the library service of Fraunhofer IESE.

cesses such as inspections, coding, or testing. This perspective is the one most commonly considered in the reviewed literature (although not stated directly) and is the usual perspective used by software practitioners [8].

Based on the aforementioned criteria, 136 references were included in the full review. After a full review, 122 references in total were considered in the results presented in this chapter. Several references (mostly books) contained well-separated sections where productivity factors were considered in separate contexts. In such cases, we considered these parts as separate references. This resulted in the final count of 142 references. For each included reference, the following information was extracted:

- Bibliographic information: title, authors, source and year of publication, etc.
- Factor selection context: implicit, explicit, cost model, productivity measure, project data repository.
- Factor selection method: expert assessment, literature survey, data analysis.
- Domain context: embedded software (Emb), management information systems (MIS), and web applications (Web).
- Size of factor set: initial, finally selected.
- Factor characteristics: name, definition, weighting (if provided).

2.2.2 Study Limitations

During the literature review, we faced several problems that may limit the validity of this study and its conclusions. First, the description of factors given in the literature is often incomplete and limited to a factor's name only. Even if both name and definition are provided, factors often differ with respect to their name, although according to the attached definition, they refer to the same phenomenon (factor).

Second, published studies often define factors representing a multidimensional phenomenon, which in other studies is actually decomposed into several separate factors. For instance, the *software performance constraints* factor is sometimes defined as time constraints, sometimes as storage constraints, and sometimes as both time and storage constraints.

Moreover, instead of factors influencing development productivity, some studies considered a specific factor's value and its specific impact on productivity. For instance, some studies identified the life cycle model applied to developing software as having an impact of productivity, while others directly pointed out incremental development as increasing overall development productivity.

Finally, factors identified implicitly within cost models (especially data-driven models) are burdened with a certain bias, since in most cases, data-driven models are built using the same commonly available (public) data repositories (e.g., [1][23][47][68][75]). In consequence, factors identified by such cost models are limited a priori to specific set of factors covered by the data repository used.

In order to moderate the impact of those problems on the study results, we undertook the following steps:

- We excluded from the analysis those models that were built on one of the publicly available project repositories and simply used all available factors. We focused on models that selected a specific subset of factors or were built on their "own", study-specific data.
- We compared factors with respect to their definition. If this was not available, we used a factor's name as its definition.

- In our study, we decomposed a complex factor into several base factors according to the factor's definition.
- We generalized factors that actually referred to a specific factor's value to a more abstract level. For instance, the "iterative development" factor actually refers to a specific value *development lifecycle model* factor – we classified this factor as "lifecycle model".
- Finally, we excluded (skipped) factors whose meaning (based on the name and/or definition provided) was not clear.

2.2.3 Demographical Information

Among the reviewed references, 33 studies identified factors directly in the context of development productivity modeling, 82 indirectly in the context of cost modeling (e.g., estimation, factor selection), 14 in the context of a project data repository (for the purpose of cost/productivity estimation/benchmarking), 9 in the context of software process improvement, 2 in the context of schedule modeling, and, finally, 2 in the context of schedule estimation (see Figure 2-1).

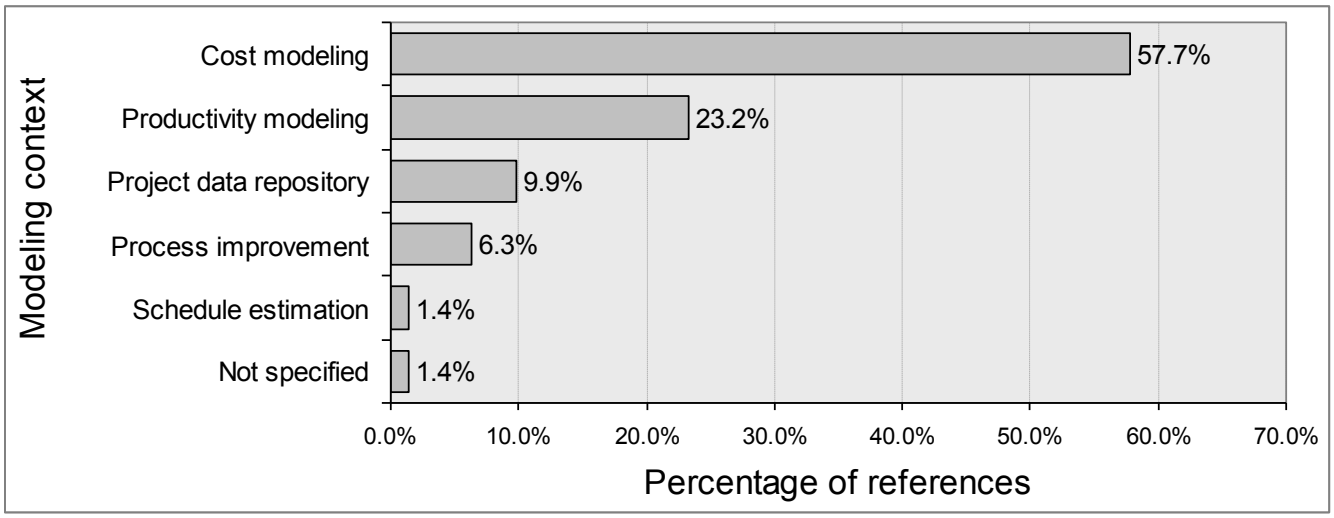

Figure 2-1 Literature distribution regarding modeling context

Regarding the domain, 27 references considered productivity factors in the context of embedded software and software systems, 50 in the context of management information systems, and 4 in the context of web applications (see Figure 2-2). The remaining references either considered multiple domains (18) or did not explicitly specify it (43).

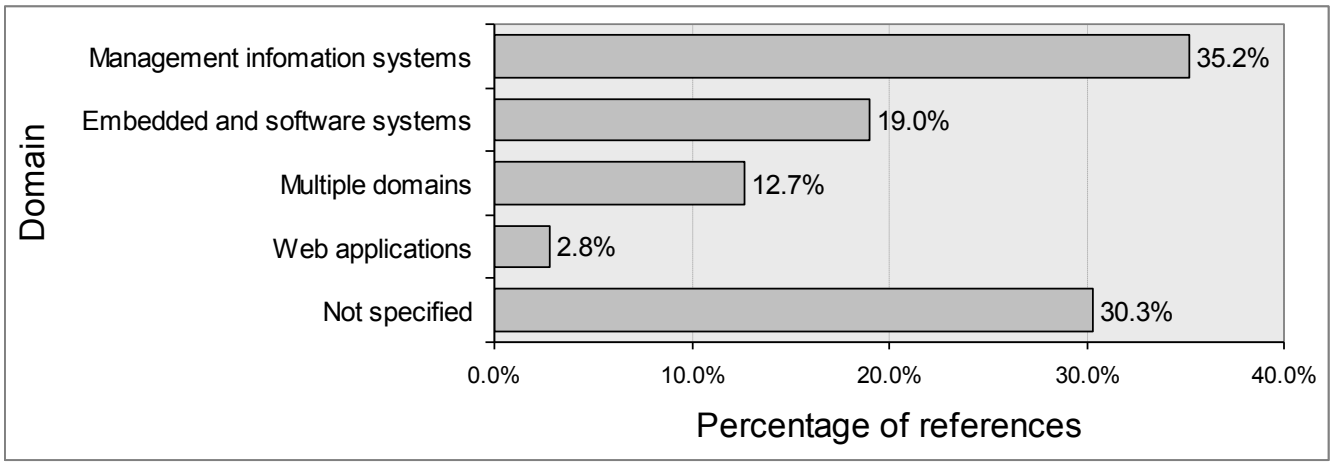

Figure 2-2 Literature distribution regarding domain

With respect to development type, 19 references analyzed productivity factors in the context of new development, 44 in the context of enhancement/maintenance, and 15 in the context of software reuse and COTS development (see Figure 2-3).

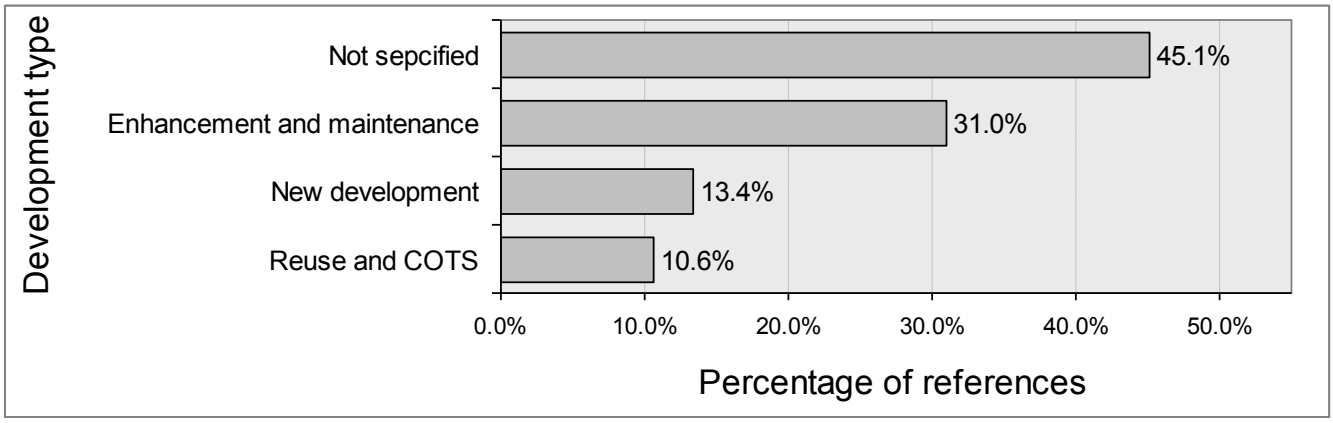

Figure 2-3 Literature distribution regarding development type

Most of the analyzed studies started with some initial set of potential productivity factors and applied various selection methods to identify only the most significant ones. The analyzed references ranged from 1 to 200 initially identified factors that were later reduced down to the 1 to 31 most significant factors.

In total, the reviewed references mentioned 246 different factors, with 64 representing abstract phenomena (e.g., *team experience and skills*) and the remaining 178 concrete characteristics (e.g., *analyst capability* or *domain experience*).

2.3 Aggregation of the Review Results

Due to space limitations, we do not provide the complete lists of factors we identified in this study. Instead, we report those factors that are most commonly considered as having the greatest impact on software development productivity.

Both in the literature as well as in the IESE studies, various methods were used to determine the significance of the considered factors. Several authors present a simple ranking of factors with respect to their

significance [27]; others used various weighting schemas to present the relative impact of each considered factor on productivity or cost [72].

We aggregated the results presented in various studies in the following procedure:

- We ranked factors with respect to the weightings they were given in the source study. The factor having the greatest weight (impact on productivity) was ranked with "1", the next one as "2", and so on. If the source study reported factors without distinguishing their impact/importance, we considered all factors as having rank "1". The disadvantage of the applied aggregation procedure is that the information regarding the "distance" between factors with respect to their weight was lost.
- We aggregated the results for each factor using two measures: (1) number of studies a given factor was considered in (*Frequency*), and (2) median factor's rank[3] over all the studies it was considered in (*Median rank*).

3. RELATED TERMINOLOGY

3.1 Context vs. Influence Factors

In practice, it is difficult to build a reliable productivity model that would be applicable across a variety of environments. Therefore, usually only a limited number of factors influencing productivity is considered within a model; the rest is kept constant and described as the so-called *context* for which the model is built and in which it is applicable. Building, for instance, a model for business application and embedded real-time software would require covering a large variety of factors that play a significant role in both domains. Alternatively, one may build simpler models for each domain separately. In that case, the factor "application domain" would be constant for each model. We would refer to factors that describe a modeling context as *context factors*. On the other hand, factors that are included in the model in order to explain productivity variance within a certain context will be called *influence factors*. Moreover, we would say that context factors determine influence factors, i.e., dependent on the project context, different factors may have a different impact on productivity and may interact with each other differently.

3.2 Classification of Influence Factors

In order to gain better visibility, we further categorized (after several other authors [51][70][110]) the identified influence factors into four groups: product-, personnel-, project-, and process factors.

Product factors cover the characteristics of software products being developed throughout all development phases. These factors refer to such products as software code, requirements, documentation, etc., and their characteristics, such as complexity, size, volatility, etc.

Personnel factors reflect the characteristics of personnel involved in the software development project. These factors usually consider the experience and capabilities of such project stakeholders as development team members (e.g., analysts, designers, programmers, project managers) as well as software users, customers, maintainers, subcontractors, etc.

Project factors regard various qualities of project management and organization, development constraints, working conditions, or staff turnover.

[3] We assume ranks to be on the equidistant ordinal scale, which allows applying median operation [82].

Process factors are concerned with the characteristics of software processes as well as methods, tools and technologies applied during a software development project. They include, for instance, the effectiveness of quality assurance, testing quality, quality of analysis and documentation methods, tool quality and usage, quality of process management, or the extent of customer participation.

4. OVERVIEW OF FACTORS PRESENTED IN LITERATURE

In total, 246 different factors were identified in the analyzed literature. This section presents the most commonly used productivity factors we found in the reviewed literature. First, we present factors that are commonly selected across all development contexts. Then, we present the most popular factors selected in the context of a specific model, development type, and domain. Additionally, factors specific for software reuse are analyzed.

Regarding the studies where project data repositories were presented, we analyzed publicly available data sets that contain more than only size and effort data (see, e.g., [85] for an overview).

4.1 Cross-Context Factors

Table 4-1 presents top 4 productivity factors and 3 context factors that were found to be the most significant ones according to all studies reviewed.

The most commonly identified factors represent complex phenomena, which might be decomposed to basic sub-factors. For each complex factor, we selected (at most) 3 most commonly identified sub-factors. *Team capability and experience*, for instance, is considered to be the most influential productivity factor. Specifically, *Programming language experience*, *Application experience & familiarity*, and *Project manager experience & skills* are those team capabilities that were most often selected in the reviewed studies.

Influence Factors	**Frequency [#references]**	**Median rank**
Team capabilities and experience	64	1
Programming language experience	*16*	*1*
Application experience & familiarity	*16*	*1*
Project manager experience & skills	*15*	*1*
Software complexity	42	1
Database size & complexity	*9*	*1*
Architecture complexity	*9*	*1*
Complexity of interface to other sys.	*8*	*1.5*
Project constraints	41	1
Schedule pressure	*43*	*1*
Decentralized/Multisite development	*9*	*1*
Tool usage & quality/effectiveness	41	1
CASE tools	*12*	*1*
Testing tools	*5*	*1*
Context factors		
Programming language	29	1
Domain	14	1
Development Type	11	1

Table 4-1 Top cross-context productivity factors

4.2 Context-specific Factors

In Table 4-2 to Table 4-4, each cell contains information in the form X/Y, where X is the number of studies where a certain factor was selected (*Frequency*) and Y means the median rank given to the factor over those studies (*Median rank*). Moreover, the most significant factors for a certain context are marked with bold font and cells containing factors that were classified as the most significant ones in two or more contexts are grey-filled. An empty cell means that a factor did not appear in a certain context at all.

For each considered context number of relevant references is given in the table header (in brackets).

4.2.1 Model-specific Factors

Table 4-2 presents the most common factors selected in the context of cost modeling (CM), productivity measurement (PM), project data repositories (DB), and studies on software process improvement (SPI).

Influence Factors	**PM (33)**	**CM (82)**	**DB (14)**	**SPI (9)**
Team capabilities & experience	***14/1***	***39/1***	***6/1***	***3/1***
Overall personnel experience	***5/1***	*4/1*	-	*1/1*
Project manager experience & skills	***3/2***	*6/7*	***3/1***	***3/1***
Task-specific expertise	***3/6***	*2/9.5*	-	-
Application experience & familiarity	*2/1*	***12/1***	*1/1*	-
Programming lang. experience	*3/9*	***11/1***	***2/1***	-
Tool experience	*1/1*	*4/1*	***2/1***	-
Teamwork capabilities	*1/1*	*2/1*	***2/1***	-
Training level	-	*4/1*	*1/1*	***2/1***
Tool usage & quality/effectiveness	***12/2***	***22/1***	*5/1*	*2/1*
CASE tools	***4/5.5***	***3/1***	*4/1*	*1/1*
Testing tools	*1/1*	***4/3.5***	-	-
Team size	***8/1***	*14/1*	*5/1*	*1/1*
Reuse	***8/1***	*9/1*	*5/1*	*2/2.5*
Reuse level	***5/1***	*7/1*	*3/1*	*2/2.5*
Quality of reused assets	***2/1***	-	-	-
Software complexity	*7/2.3*	***25/1***	***8/1***	*1/1*
Architecture complexity	*2/6.5*	***6/1***	*1/1*	-
Complexity of interface to other systems	*1/1*	***6/2***	*1/1*	-
Database size & complexity	*1/1*	***4/10***	***4/1***	-
Code complexity	*1/1*	*2/1*	***4/1***	-
Team organization	*6/2*	***21/1***	*3/1*	***3/1***
Team cohesion/communication	*3/3*	***14/1***	-	*1/3*
Staff turnover	-	***11/1***	*2/1*	***2/1.5***
Team structure	*4/1*	***4/3.5***	*1/1*	***2/1***
Project constraints	*9/3*	***21/3***	***6/1***	***3/1***
Schedule pressure	*9/1*	***16/5***	***5/1***	***2/1***
Decentralized development	*1/16*	***5/7***	***2/1***	*1/1*
Process maturity & stability	2/1	7/1	-	**4/1**
Methods usage	7/6	16/1	5/1	**3/1**
Reviews & inspections	*2/11*	*4/1*	*2/1*	***2/1***

Testing	*1/5*	*2/1*	*1/1*	***2/1***
Requirements management	-	*2/1*	*1/1*	***2/1***
Context Factors				
Programming language	**8/1**	**12/1**	**8/1**	1/1
Life cycle model	**2/1**	3/1	**3/1**	1/1
Domain	**2/4.5**	**7/1**	**4/1**	1/1
Development type	-	**7/1**	**3/1**	1/1
Target Platform	1/1	3/1	**3/1**	-

Table 4-2 Top model-specific productivity factors

4.2.2 Development-Type-specific Factors

Table 4-3 presents the most common productivity factors selected in the context of *new development* (Nd) and *maintenance/enhancement* projects (Enh/Mtc). We considered maintenance and enhancement together because the considered references did not make a clear difference between those two types of development.

Influence Factors	**Nd (19)**	**Enh/Mtc (43)**
Team capabilities & experience	**8/1**	**26/1**
Task-specific experience	***3/11***	*1/5*
Application experience & familiarity	***1/1***	***11/1***
Programmer capability	***1/1***	*5/3*
Programming language experience	-	***10/1.5***
Analyst capabilities	-	***6/3***
Project constraints	**7/6**	**17/1**
Schedule pressure	***7/5***	***13/2***
Distributed/multisite development	***3/8***	***4/1***
Reuse	**5/1**	11/4
Reuse level	***4/1***	*7/4*
Quality of reusable assets	***1/1***	*1/1*
Management quality & style	**5/1**	5/8
Team motivation & commitment	**5/1**	4/4.5
Product complexity	5/7	**17/2**
Interface complexity to hardware & software	-	***6/2***
Architecture complexity	*1/1*	***6/2***
Database size and complexity	-	***4/10***
Logical problem complexity	-	***3/1***
Required software quality	4/2	**15/4**
Required reliability	*4/2*	***13/4***
Required maintainability	-	***3/4***
Tool usage	3/1	**17/2**
Testing tools	-	***3/5***
Tool quality and effectiveness	*1/6*	***2/1.5***
Method usage	2/4.3	**13/4**
Review & Inspections	*1/8*	***4/1***
Testing	-	***3/1***
Context factors		
Programming language	**4/1**	**10/1**

Target platform	**1/1**	3/1
Domain	-	**8/3**
Development type	-	**6/1**

Table 4-3 Top development-type-specific productivity factors

4.2.3 Domain-specific Factors

Table 4-4 presents the most common productivity factors selected in the context of specific software domains: embedded systems (Emb), management and information systems (MIS), and Web applications (Web).

Influence Factors	**Emb (27)**	**MIS (50)**	**Web (4)**
Team capabilities & experience	**14/1**	**20/1**	**3/3**
Programming language experience	***5/1***	*4/3*	*1/3*
Training level	***3/1***	*1/1*	-
Application experience & familiarity	*1/1*	***9/1***	-
Programmer capability	*1/2*	***6/3.5***	-
Design experience	-	*1/3*	***1/1***
IT technology experience	*1/1*	*1/1*	***1/1***
Team size	**8/1**	**9/1**	1/3
Reuse	3/1	**9/1**	1/5
Reuse level	*2/3.5*	***6/1***	*1/5*
Tools usage & quality	**12/1**	**9/1**	**3/2**
CASE tools	***3/1***	***4/1***	-
Methods usage	**12/1**	8/1.5	-
Reviews & inspections	***4/1***	*1/1*	-
Testing methods	***2/3***	*1/1*	***1/2***
Modern programming practices	***2/6***	*2/5*	-
Project constraints	**9/1**	**16/2.2**	-
Schedule pressure	***7/1***	***12/3***	-
Distributed/multisite development	*1/1*	***5/7***	-
Requirements quality	**8/1**	9/2	1/2
Requirements quality	***4/1***	-	-
Requirements volatility	***4/1***	*8/2.5*	*1/2*
Required software quality	**8/1**	**12/3.5**	2/4.5
Required sw. reliability	***7/1***	***10/4***	-
Required sw. maintainability	-	***4/3.5***	-
Software complexity	3/1	**17/1**	**3/3**
Database size & complexity	-	***7/1***	-
Source code complexity	*1/1*	***4/1***	-
Complexity of interface to other systems	*3/1*	*2/2.5*	***1/3***
Architecture complexity	*3/1*	*3/1*	***1/1***
Context factors			
Programming language	**7/1**	**10/1**	**1/1**
Domain	**4/1**	4/2	-
Target platform	2/1	**4/1**	-
Life cycle model	2/1	3/1	-
Development type	1/1	**4/1**	**1/1**

Business sector	1/14	**4/1**	-

Table 4-4 Top productivity factors used to model software cost

4.3 Reuse-specific Factors

From among the reviewed references, 15 focused specifically on reuse-oriented software development, i.e., development with and for reuse (including COTS – commercial of-the-shelf components). Table 4-5 summarizes the most common factors influencing development productivity in that context.

Factor	Frequency [#references]	Median Weight
Quality of reusable assets	12	1
Maturity	*7*	*1*
Volatility	*5*	*1*
Reliability	*3*	*1*
Asset complexity	9	1
Interface complexity	*5*	*1*
Architecture complexity	*4*	*1*
Team capabilities & experience	6	1
Integrator experience & skills	*3*	*1*
Integrator exp. with the asset	*3*	*1*
Integrator exp. with COTS process	*3*	*1*
Product support (from supplier)	6	1
Supplier support	*6*	*1*
Provided training	*5*	*1*
Required software quality	4	1
Required reliability	*3*	*1*
Required performance	*3*	*1*
Context factors		
Life cycle model	2	1
Programming language	1	1
Domain	1	1

Table 4-5 Top reuse-specific productivity factors

4.4 Summary of Literature Review

The review of the software engineering publications presented here shows that software development productivity still depends on the capabilities of people and tools involved (Figure 4-1).

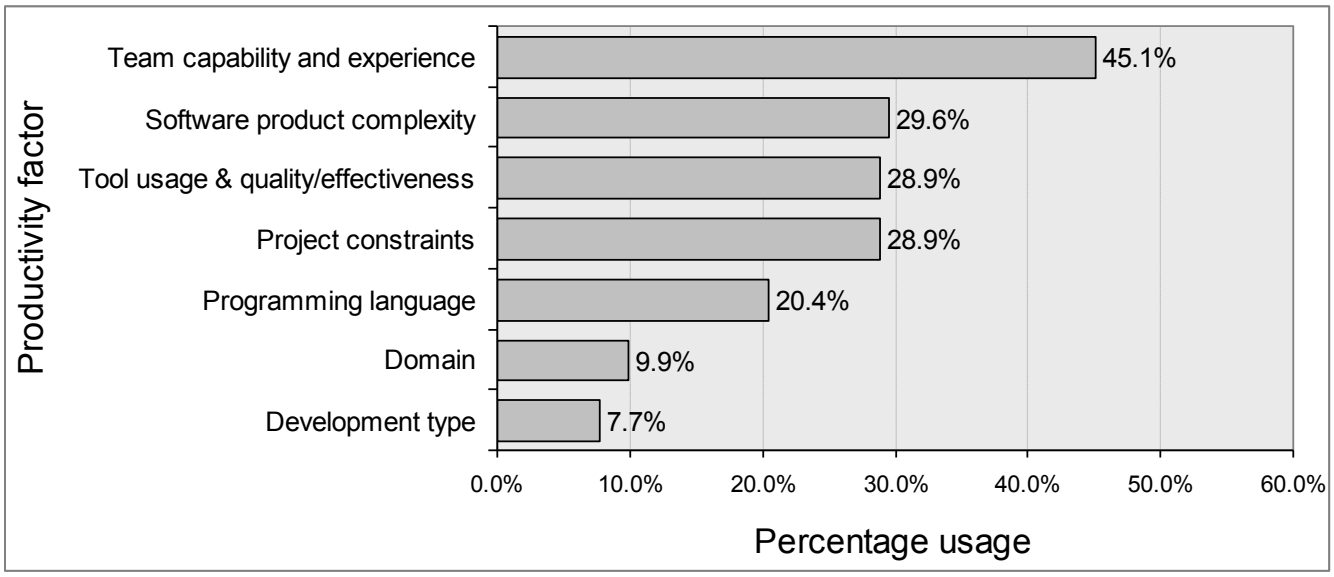

Figure 4-1 Literature review: Most common productivity factors

The productivity factors selected by researchers and software practitioners confirm requirements specification, coding, and testing as the traditionally acknowledged essential phases for the success of the whole development process. Moreover, the high importance of *project constraints* and *project manager's skills* suggests project management as another key factor for project success. Finally, as might have been expected, the internal (architecture, data) and external (interfaces) complexity of software is a significant determinant of development productivity.

Yet, software complexity and programming language are clearly factors preferred in the context of project data repositories. This most probably reflects a common intuition of repository designers that those factors have a significant impact on development productivity and numerous software qualities (e.g., reliability, maintainability).

As already mentioned in the introduction, the importance of a certain productivity factor varies depending on the project context. The skills of software programmers and analysts, for instance, seem to play a more important role in enhancement/maintenance projects. Similarly, tool/method usage seems to be less important in new development projects. On the other hand, software development that is not a continuation of a previous product/release seems to significantly rely on the quality of project management and team motivation.

The results presented in the literature support the claim made by Fenton and Pfleeger [51] who suggest that software complexity might have a positive impact on productivity in the context of new development (and thus is not perceived as a factor worth considering) and a negative impact in case of maintenance and enhancement.

What might be quite surprising is that the domain is not considered in the new development project at all.

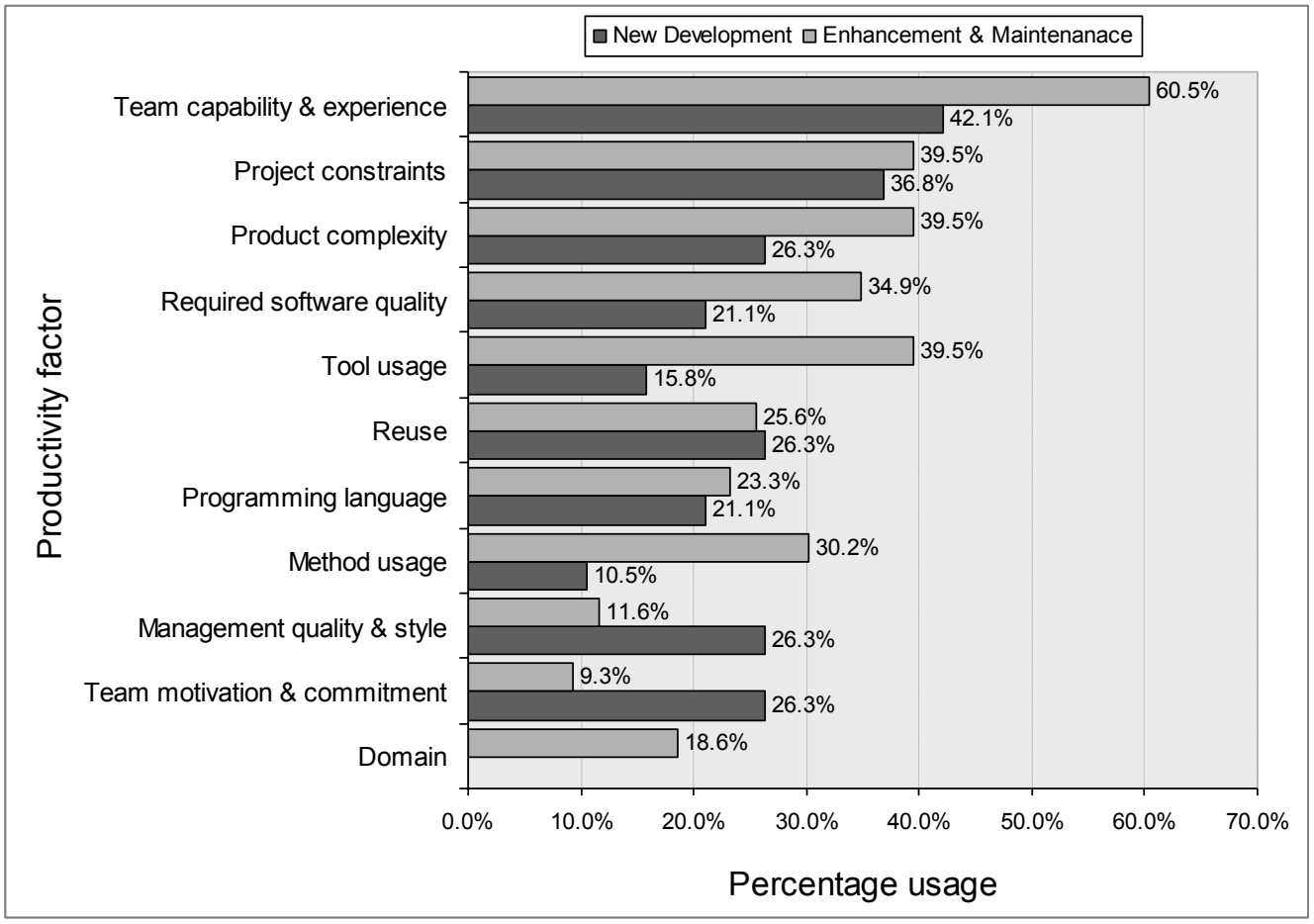

Figure 4-2 Literature review: Development-specific factors

Regarding software domain-specific factors, there are several significant differences between factors playing an important role in the embedded and MIS domains. The productivity of embedded software development depends more on *tools and methods used*, whereas that of MIS depends more on the *product complexity*.

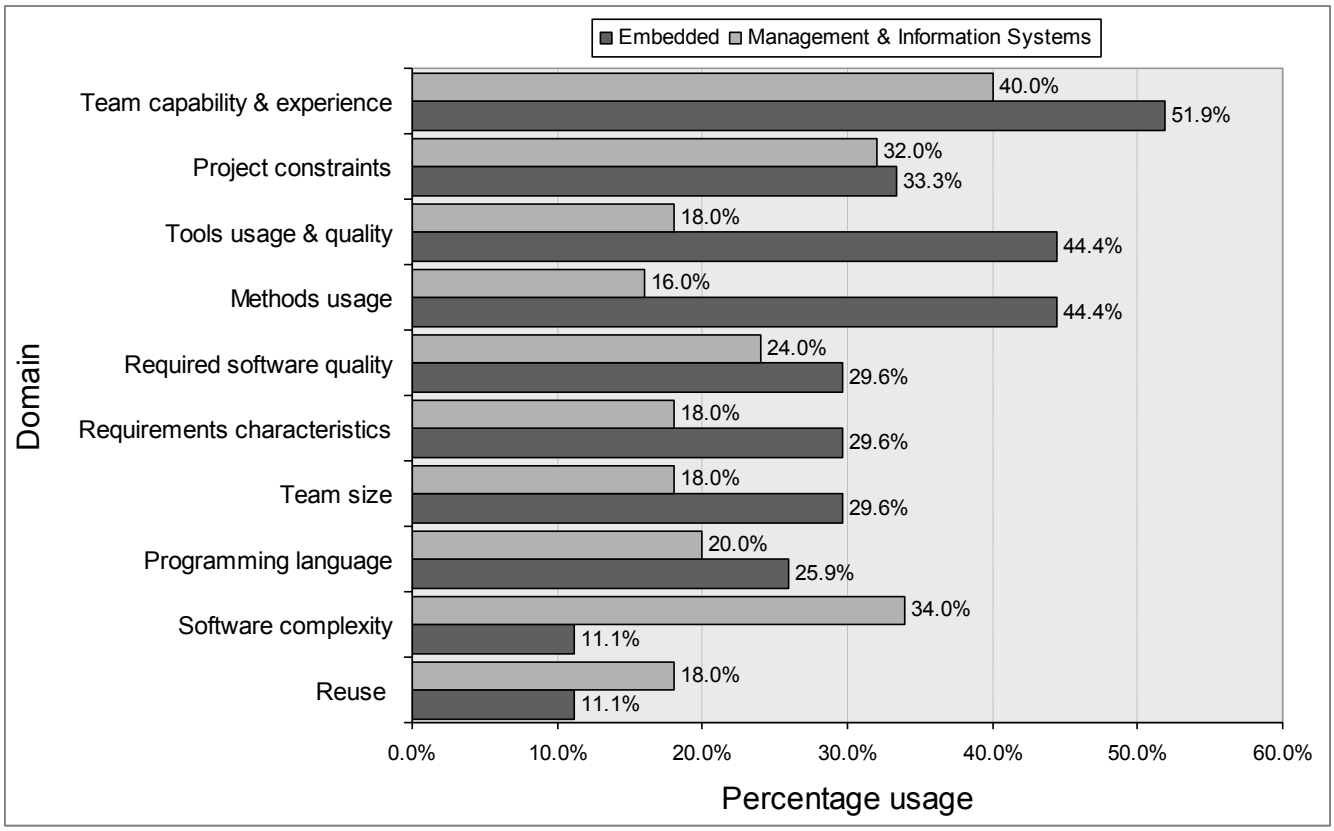

Figure 4-3 Literature review: Domain-specific factors

Finally, *reuse* is not as significant a productivity factor as commonly believed. Less than 17% of publications mention reuse as having a significant influence on development productivity. This should not be a surprise, however, if we consider the complex nature of the reuse process and the numerous factors determining its success (Figure 4-4).

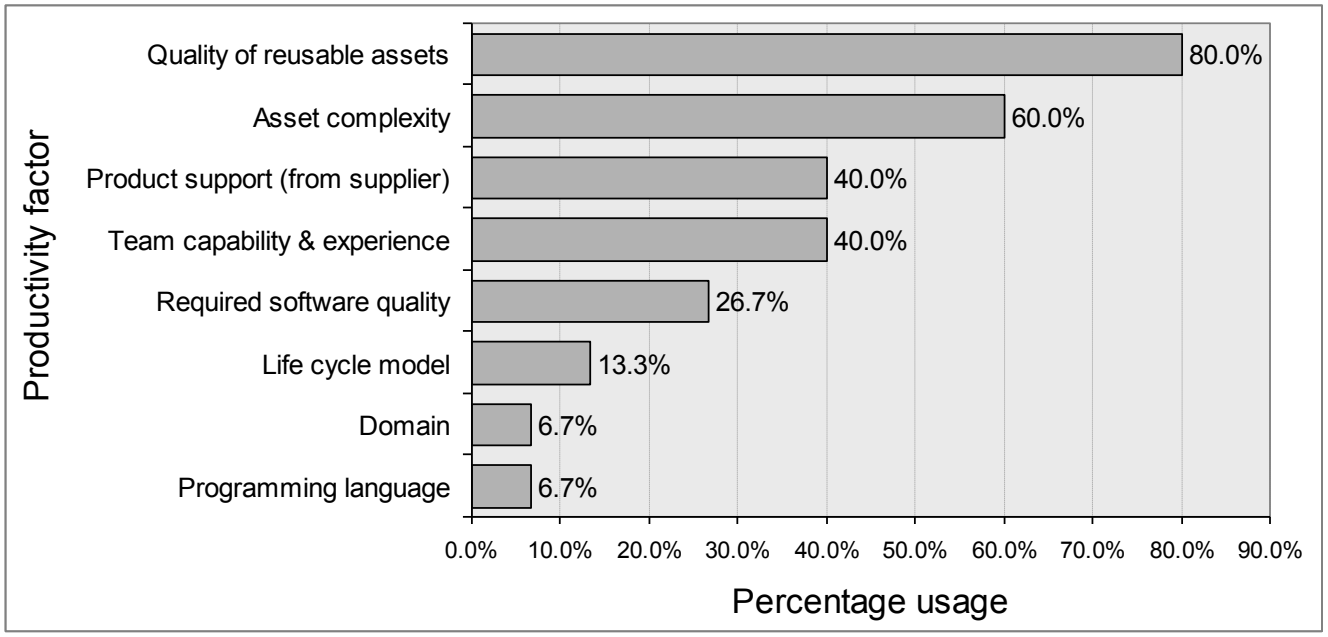

Figure 4-4 Literature review: Reuse success factors

According to the reviewed studies, the complexity and quality of reusable assets are key success factors for software reuse. Furthermore, the support of the asset's supplier and the capabilities of development team (for integrating the asset into the developed product) significantly influence the potential benefits/losses of reuse (see Section 6.2.4 for a comprehensive overview of key reuse success factors).

5. OVERVIEW OF FACTORS INDICATED BY INDUSTRIAL EXPERIENCES

This section summarizes experiences regarding the most commonly used productivity factors indicated by our industrial experiences gained in recent years at Fraunhofer IESE. The studies summarized here are subject to a non-disclosure agreement, thus only aggregated results without any details on specific companies are presented.

5.1 Demographics

The IESE studies considered in this section include:

- 13 industrial projects on cost and productivity modeling performed for international software organizations in Europe (mostly), Japan, India and Australia.
- 4 international workshops on software cost and productivity modeling, which took place in the years 2005-2006 in Germany and Japan. The workshop participants came from both academic (universities, research institute) and industrial (software and system) contexts.
- 8 worldwide surveys on cost and productivity modeling. This includes one cross-company survey where we talked to a single representative of various software organizations, and 7 surveys where we talked to a number of software engineers within a single company.

The studies considered here covered a variety of software domains (Figure 5-1) and development types (Figure 5-2).

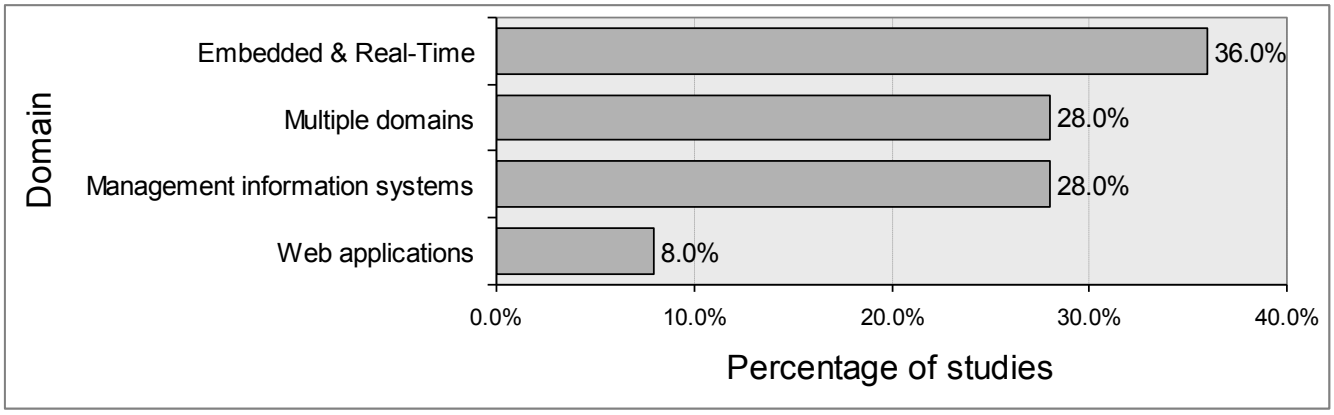

Figure 5-1 Industrial experiences: Application domains considered

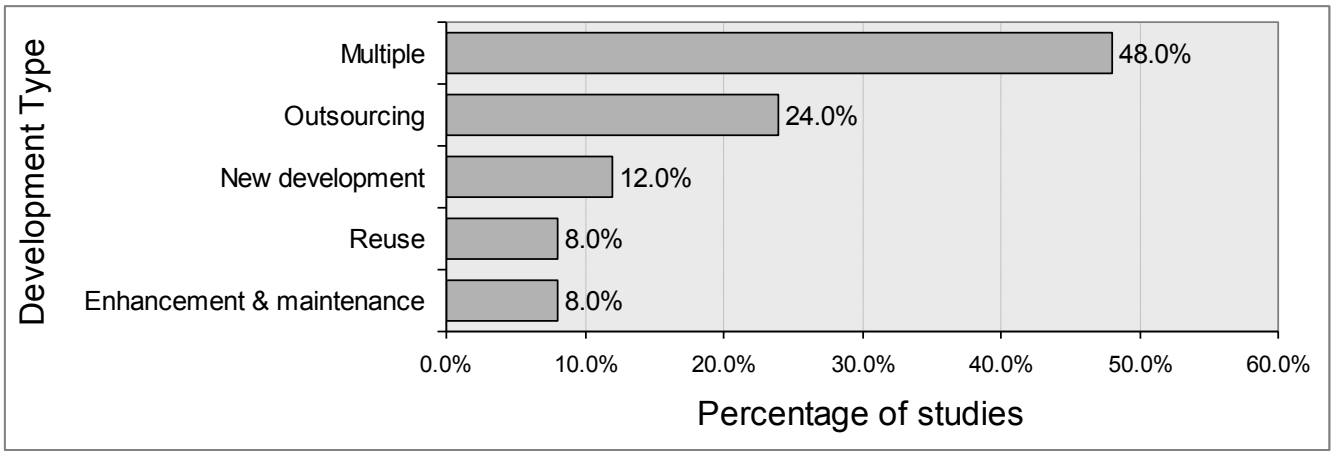

Figure 5-2 Industrial experiences: Development types considered

In total, we identified 167 different factors, with 31 of them representing complex phenomena and 136 basic concepts (sub-factors).

5.2 Cross-Context Factors

Table 5-1 and Figure 5-3 present these productivity factors that were selected most often in the context of all IESE studies (commercial projects, surveys, and workshops).

Factor	Frequency [#references]	Median rank
Requirements quality	23	1
Requirements volatility	*20*	*1*
Requirements novelty	*11*	*1*
Team capabilities & experience	23	3
Project manager experience & skills	*10*	*4.5*
Programming lang. experience	*10*	*17*
Teamwork & communication skills	*9*	*3*
Domain experience & knowledge	*9*	*5*
Project constraints	20	4.6
Schedule pressure	*13*	*2*
Distributed/multisite development	*11*	*6*
Customer involvement	18	2
Method usage & quality	18	4.3
Requirements management	*10*	*2.5*
Reviews & inspections	*7*	*5*
Required software quality	18	8.5
Required software reliability	*16*	*4*
Required software maintainability	*10*	*15*
Context factors		
Life cycle model	8	3
Development type	3	5
Domain	2	1.5

Table 5-1 Industrial experiences: Cross-context productivity factors

Similarly to published studies, *development team capabilities*, *project constraints*, and *method usage*, were considered as the most significant factors influencing software development productivity. Unlike other practitioners, software engineers involved in IESE undertakings selected *requirements novelty and stability*, *customer involvement*, and *required software reliability and maintainability* as high-importance productivity factors. High importance of requirements volatility and customer involvement might be connected to the fact that most of the IESE studies regarded outsourcing projects where stable requirements and communication between software supplier and contractor are essential for project success.

The fact that *programming language* does not occur at all as a context factor results from the homogenous context of most IESE studies (most of them regard a single business unit or group where a single programming language is in use).

5.3 Context-specific Factors

In this section, we look at the variances in selected productivity factors across different contexts they were selected in.

Each cell of Table 5-2 and Table 5-4 contains information in the form X/Y, where X is the number of studies where a certain factor was selected (*Frequency*) and Y means the median rank given to the factor over those studies (*Median rank*). Moreover, the most significant factors for a certain context are marked with bold font and cells containing factors that were classified as the most significant ones in two or more contexts are grey-filled. An empty cell means that a factor did not appear in a certain context at all.

For each context considered, the number of relevant references is given in the table header (in brackets).

5.3.1 Study-specific Factors

In Table 5-2, the most commonly selected factors across various study types are presented.

Notice that practically no context factors were considered in the context of commercial projects, which in fact, took place in a homogeneous context where all usually significant context factors were constant (thus had no factual impact on productivity).

Moreover, product qualities and constraints such as reliability, maintainability, and performance (execution time and storage) are also mentioned only in the context of specific commercial projects.

Factor	C&P (13)	Srv (8)	Wsk (4)
Requirements quality	**13/3**	**8/1**	2/1
Requirements volatility	***12/1***	***6/1***	*2/1*
Requirements novelty	***7/9***	***3/1***	*1/1*
Required software quality	**13/13.5**	4/2	1/1
Required software reliability	***12/5***	*3/3*	*1/1*
Required sw. maintainability	***9/16***	*1/1*	-
Required software usability	***7/19***	-	-
Project constraints	**12/7.5**	5/3	**3/1**
Schedule pressure	***6/2.5***	*5/3*	***2/1***
Distributed/multisite development	***5/14***	*4/3.5*	***2/1***
Budget constraints	-	-	***2/1***
Team capability & experience	**12/7.3**	**8/1**	**3/1**

Programming language experience	***8/20.5***	*1/15*	*1/1*
Project manager experience & skills	***7/8***	*2/4.5*	*1/1*
Platform (virtual machine) experience	***5/14***	-	-
Teamwork & communication skills	*3/4*	***6/2***	-
Domain experience & knowledge	*4/18.5*	***4/3***	*1/1*
Training level	*3/19*	*1/4*	***2/1***
Software product constraints	**11/12**	2/8	1/1
Execution time constraints	***11/11***	*2/7*	-
Storage constraints	***6/18.5***	*1/1*	-
Customer/user involvement	10/2	**7/2**	1/1
Method usage	10/8.5	**7/3**	1/1
Reviews & inspections	*3/9*	***4/1.5***	-
Requirements management	*6/1.5*	***4/3***	-
Tool usage & quality	9/10	**6/4.7**	1/1
Testing tools	*1/35*	***2/3***	-
Project management tools	*1/42*	***2/10.5***	-
Reuse	10/14	4/2	**3/1**
Reuse level	*6/13.5*	*3/3*	***2/1***
Required software reusability	*6/12*	*1/18*	***2/3***
Context factors			
Life cycle model	**2/18**	**3/5**	**3/1**
Domain	-	**2/1.5**	-
Development type	-	**2/6**	**1/1**

Table 5-2 Industrial experiences: Study-specific productivity factors

5.3.2 Development-Type-specific Factors

For most of the studies considered here, the development type was either not clear (not explicitly stated) or productivity factors were given as valid for multiple development types. Since there is too little data to analyze productivity factors for most of the development types considered, we only take a closer look at factors that are characteristic for organizations outsourcing their software development (Table 5-3).

Factor	Frequency [#references]	Median Weight
Requirements quality	6	6.2
Requirements volatility	*6*	*1.5*
Requirements novelty	*5*	*11*
Customer/user involvement	6	4
Project constraints	6	8.5
Distributed/multisite development	*3*	*14*
Concurrent hardware development	*3*	*12*
Legal constraints	*3*	*30*
Software complexity	6	12.5
Database size & complexity	*6*	*22*
Complexity of device-dependent operations	*2*	*8,5*
Complexity of control operations	*2*	*9*
Context factors		
Life cycle model	1	18

Table 5-3 Industrial experiences: Productivity factors in the outsourcing context

It might be no surprise that in outsourcing projects, *requirements quality* and *customer involvement* are key factors driving development productivity. Stable requirements and communication with the software customer (especially during early development phases) are commonly believed to have crucial impact on the success of outsourcing projects.

5.3.3 Domain-specific Factors

Unfortunately, most of the IESE studies considered either regarded multiple domains or the domain was not clearly (explicitly) specified. Therefore, we analyzed only factors specific for the embedded systems (Emb) and management and information systems (MIS) domains, for which a reasonable number of inputs were available.

Similar to what we found in the related literature, the *usage of tools* (especially testing) is characteristic of the embedded domain. Regarding *method usage*, different methods play an important role in the embedded and MIS domains. Use of early quality assurance methods (reviews & inspections) is regarded as having a significant impact on productivity in the MIS domain, whereas use of late methods, such as testing, counts more in embedded software development.

Requirements management as well as configuration and change management activities are significant productivity factors only in the MIS domain. Software practitioners do not relate them to productivity variance in the embedded domain because they are usually consistently applied across development projects. According to our observation, however, the effectiveness of those activities varies widely and therefore should be considered as a significant influence factor.

Factor	**Em (9)**	**MIS (7)**
Requirements quality	**9/5**	**7/1**
Requirements volatility	***9/1***	***7/1***
Requirements novelty	***6/10***	-
Requirements complexity	-	***2/1***
Software complexity	**9/5**	5/9
Architecture complexity	***3/7***	-
Database size & complexity	***3/22***	*2/16.5*
Team capabilities and experience	**9/8.4**	**7/3**
Programming language experience	***7/19***	
Project manager experience & skills	***5/9***	***3/3***
Teamwork & communication skills	*2/1*	***5/3***
Customer/user involvement	**8/2**	**7/2**
Involvement in design reviews	***2/19.5***	-
Tool usage & quality	**8/9**	3/15
Testing tools	***2/18***	-
Method Usage	**8/9.6**	**7/3**
Requirements management	3/3	**7/2**
Reviews & inspections	2/6.5	**4/7**
Testing methods	4/5.5	2/8
Documentation methods	5/20	-
Configuration management & change control	2/22.5	**3/4**
Project constraints	7/9	**6/3.7**
Schedule pressure	3/9	**5/2**

Distributed/multisite development	3/14	**4/8.5**
Context factors		
Life cycle model	**3/14**	**1/9**
Domain	**1/1**	**1/2**
Application type	-	**1/2**

Table 5-4 Industrial experiences: Domain-specific productivity factors

5.4 Summary of Industrial Experiences

The productivity factors observed in the most recent IESE studies do not differ much from those indicated in the reviewed literature. *Capabilities of development team*, *project constraints*, and *methods usage* are the main impact factors. The outsourcing character of the projects considered in the majority of the IESE studies gave priority to some additional factors such as *requirements quality* (volatility, complexity, and novelty), *required product quality* (reliability, maintainability, performance), and *customer involvement*.

There were slight differences between the factors considered across various domains. As in the reviewed literature, *tool and method usage* were regarded as more significant in the embedded domain.

The IESE studies considered referred to a rather narrow context (organization, business unit, group, etc.), where such characteristics as domain and programming language were quite homogeneous. That is most probably why common context factors from the literature (see Section 4.1) do not actually count in the context of IESE studies. Constant factors do not introduce any variance across projects and thus do not explain productivity variance, which in practice makes their consideration useless.

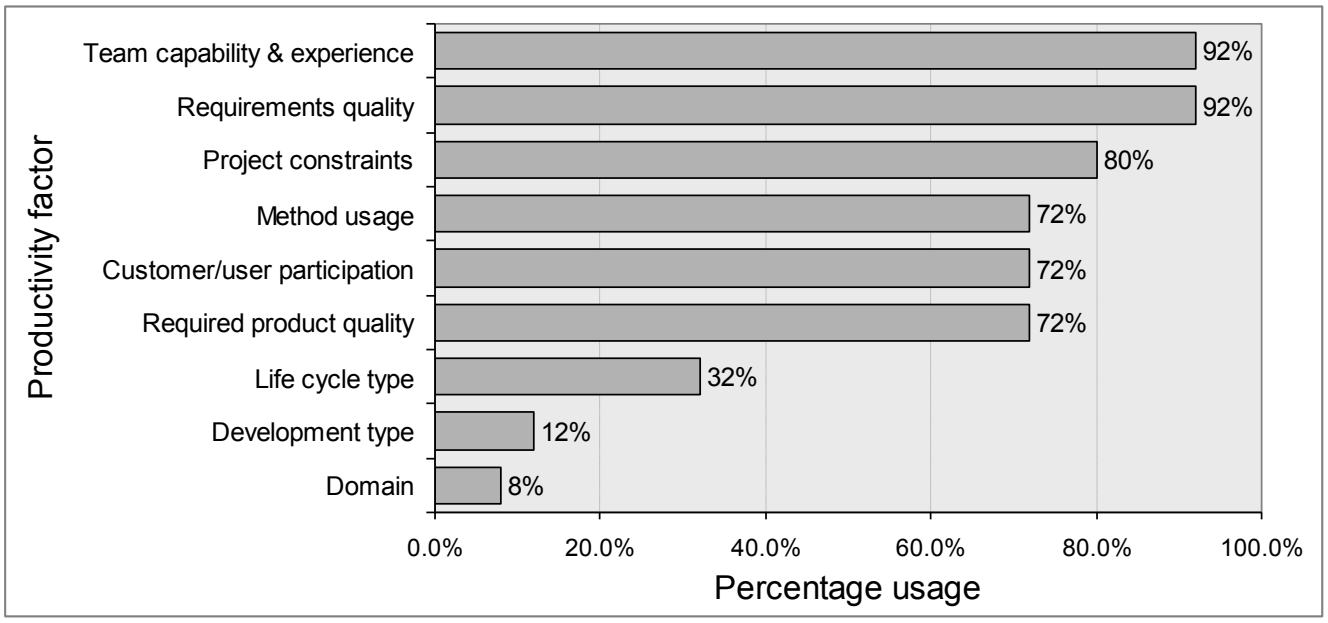

Figure 5-3 Industrial experiences: Cross-context factors

One quite surprising observation is that the implementation of early quality assurance techniques such as reviews and inspections does not seem to have a deciding impact on development productivity. In only 6 out of 25 studies, this factor was stated explicitly as having a significant impact on productivity. At the same time, our practical experiences indicate that those activities are usually ineffective or are not ap-

plied at all. This contradicts the common belief that early quality assurance activities contribute significantly to the improvement of development productivity (through decreased testing and rework effort).

Another interesting observation is that while *requirements volatility* is considered to have a significant impact on development productivity, *requirements management* is not consistently regarded as having such an impact. It is, however, believed that proper quality management may moderate the negative influence of unstable requirements on development productivity.

Finally, an interesting and, at the same time, surprising observation we made is that very mature organizations (e.g., 2 organizations at CMMI L-5 and one ISO-9001 certified) consider as very significant factors such factors as *Clarity of development team roles and responsibilities,* which are actually common for rather immature organizations.

6. DETAILED COMMENTS ON SELECTED PRODUCTIVITY FACTORS

This section presents a brief overview of publications presenting in-depth investigations on rationales underlying the influence of certain factors on software development productivity as well as dependencies between various factors. The summary presented here includes empirical research studies as well as industrial experiences.

6.1 Comments on Selected Context Factors

This section presents an overview of literature regarding the experiences with context factors, i.e., factors that are used to limit the context of productivity modeling and analysis (see Section 3.1).

6.1.1 Programming language

The analysis of the factors presented in the reviewed literature showed that programming language is the most common context factor (Table 4-1). The impact of a programming language on development productivity is considered to be so large that some authors present average development productivities for certain languages, independent of other potential factors [76]. Moreover, a single organization or business unit usually uses a limited number of distinct languages. Therefore, considering this factor when analyzing productivity within a single organization usually makes no sense. This conclusion is confirmed several data analysis studies (e.g., [89][125]) where programming language has significant impact on productivity when analyzed on cross-organization data whereas no influence was observed on organization-specific data.

6.1.2 Application domain

According to the literature review presented (Table 4-1), the application domain is considered as the second most significant context factor.

The studies presented in the literature provide evidence not only that factors influencing productivity vary across different application domains, but also that, in principle, the magnitude of productivity alone varies across different domains. Putnam and Myers [106], for example, analyzed the QSM data repository [107] and found out that there is a common set of factors influencing software process productivity. They also found out that the analyzed projects tend to cluster around a certain discrete value of productivity[4], and that except for a limited number of projects (outliers), each cluster actually represents a certain application domain. Therefore, although an exact productivity measurement would require considering other influence factors within a certain context (cluster), general conclusions about productivity

[4] The standard deviation of process productivity within clusters ranged from ± 1to ± 4.

within a certain context can already be drawn. The authors provided evidence that projects in a real-time domain tend to be less productive, in general, than in the business systems domain. Jones combines productivity variance across domains with different levels of project documentation generally required in each domain. He observed [70] that the level of documenting varies widely across various software domains. The productivity of military projects, for instance, suffers due to the extremely high level of documentation required (at least 3 times more documentation per function point is required than in MIS and Software Systems domains) [70].

6.1.3 Development type

Development type is another important factor that determines how other project characteristics influence development productivity.

Fenton and Pfleeger [51] suggest, for instance, that software complexity might have a positive impact on productivity in the context of new development and a negative one in case of maintenance and enhancement. The development team is able, for example, to produce more output (higher productivity) even though the output is badly structured (high complexity). Yet, one may expect that if producing badly-structured output might be quite productive, then maintaining it might be quite difficult and thus unproductive. This could be, for instance, observed well in the context of software reuse, where the required quality and documentation of reusable artifacts have a significant impact on the productivity of their development (negative impact) and reuse (positive impact).

In the context of maintenance projects, the purpose of software change has a significant impact on productivity. Bug fixes, for instance, are found to be more difficult (and therefore less productive) than comparably sized additions of new code by approximately a factor of 1.8 [55]. Bug fixes tend to be more difficult than additions of new code even when additions are significantly larger.

Corrective maintenance (bug fixes), however, seems to be much more productive than perfective maintenance (enhancements) [16]. This supports the common intuition that error corrections usually consist of a number of small isolated changes, while enhancements include larger changes to the functionality of the system. In both types of changes, the extent of coupling between modified software units may amplify or moderate the impact of change type on productivity. Changing a certain software component requires considering the impact of the change on all coupled components. This usually includes reviewing, modifying, and re-testing related (coupled) components. Therefore, the more coupled components the more additional work is required (lower productivity).

The time spam between software release and change (so-called “aging” or “decay” effect) is also considered as a significant productivity factor in enhancement and maintenance projects. Graves [55], for instance, confirms the findings of several earlier works. He provides statistically significant evidence that a one-year delay in change would cost 20% more effort than similar change done earlier.

Finally, maintenance productivity has traditionally been influenced by the personal capabilities of software maintainers. It was, for example, found that one developer may require 2.75 times as much effort as another developer to perform comparable changes [55]. Yet, it is not always clear exactly which personnel characteristics are determinant here (overall experience, domain experience, experience with changed software, extent of parallel work, etc.).

6.1.4 Process maturity

Process maturity is rarely identified directly as a factor influencing development productivity. Yet, numerous companies use project productivity as the indirect measure of software process maturity and/or use productivity improvement as a synonym of process improvement. Rico [108] reports, for instance,

that 22% of all measures applied for the purpose of software process improvement (SPI) are productivity measures. Another 29% are development effort, cost, and cycle time measures, which, in practice, are strongly related to development productivity.

A common belief that pushes software organizations towards process improvement is that high-maturity organizations (e.g., as measured according to the CMMI model [38]) are characterized by high-productivity processes[5] [102][104][24][48]. In fact, there are several studies that provide quantitative evidence of that belief. Diaz and King [48] report, for instance, that moving a certain software organization from CMM level 2 to level 5 in the years 1997-2001 consistently increased project productivity by the factor 2.8. A threefold increase in productivity within organizations approaching higher CMM levels has been confirmed by several other software companies worldwide [102]. Harter et al [60] investigate the relationships between process maturity measured on the CMM scale, development cycle time, and effort for 30 software products created by a major IT company over a period of 12 years. They found that in the average values for process maturity and software quality, a 1% improvement in process maturity leads to a 0.32% net reduction in cycle time, and a 0.17% net reduction in development effort (taking into account the positive direct effects and the negative indirect effects through quality).

Yet, according to other studies (e.g., [39][62]), overall organization maturity can probably not be considered as a significant factor influencing productivity. One may say that high maturity entails a certain level of project characteristics (e.g., CMMI key practices) positively influencing productivity; however, which characteristics influence productivity and to which extent varies most probably between different maturity levels. In that sense, process maturity should be considered as a context rather than as an influence factor. Influence factors should then refer to single key practices rather than to whole maturity levels.

Overall maturity improvement should be selected if the main objective is a long-term one, e.g., high-quality products delivered on time and within budget. In that case, increased productivity is not the most important benefit obtained from improved maturity. The more important effects of increased maturity are stable processes, which may, in turn, facilitate the achievement of short-term objectives, such as effective productivity improvement (Figure 6-1). One must be aware that although increasing the maturity of a process will not hurt productivity in the long-term perspective, it may hurt it during the transition to higher maturity levels. It has been commonly observed (e.g., [57]) that the introduction of procedures, new tools, and methods is, in the short-term perspective, detrimental to productivity (so-called *learning effect*). Boehm and Sullivan [25], for instance, illustrate productivity behavior when introducing new technologies (Figure 6-1).

[5] Putnam [104] analyzed the QSM database and showed that there seems to be a strong correlation between an organization's CMM level and its productivity index.

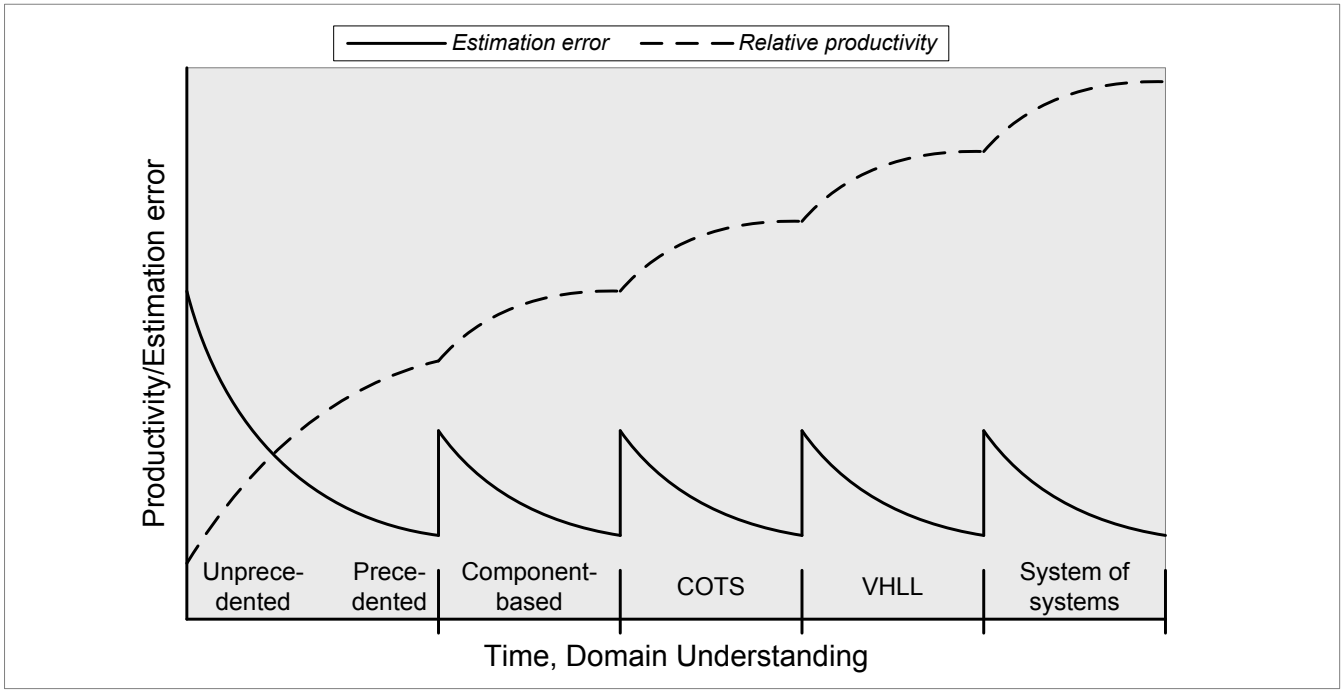

Figure 6-1 Short-term effect on productivity of introducing new technology [25]

This learning effect can be moderated by introducing a so-called *delta team* consisting of very skilled personnel who are is able to alleviate the short-term, negative effect on productivity of implementing certain key process areas (KPA's). This is, however, nothing else than preventing the productivity decrease caused by investments on implementing certain KPAs by improving factors that have proven to have a significant positive impact on productivity (in this case, team capabilities). Benefits from process improvement and from introducing new technologies can also be gained faster by sharing experiences and offering appropriate training and management support [121][124].

An alternative way of moderating the short-term, negative impact of process improvement on productivity would be to first implement KPAs that have a short-term, positive impact on productivity (e.g., team capabilities, personnel continuity) in order to offset the negative impact of implementing other KPAs. Example practices (KPAs) that have proven to make the greatest contribution to organizational maturity (highest benefits) can be found in [102].

Yet, a certain maturity level (optionally confirmed by a respective certification) is rather a "side effect" (a consequence) of process improvement (e.g., driven by a high-productivity objective) rather than an objective for its own sake – which is the right sequence [5].

Finally, the maturity of measurement processes should be the subject of special attention since inconsistent measurements usually lead to misleading conclusions with respect to productivity and its influencing factors. In that sense, one may say that mature and rigorous measurement processes have a significant impact on productivity [26]. Boehm and Sullivan [25] claim that there is about a ±15% range of variation in effort estimates between projects and organizations due to the counting rules for data. Niessink and van Vlied [96] observed, in turn, that the existence of a consistently applied (repeatable) process is an important prerequisite for a successful measurement program. If, for instance, a process exists in multiple variants, it is important to identify those factors that differentiate various variants and to know which variant is applied when.

6.1.5 Development life cycle

Only a few studies selected the development life cycle model as a significant factor influencing software productivity. This factor is, however, usually addressed indirectly as a context factor. Existing empirical studies and in the field experience (e.g., [117]) confirm, for instance, the common belief that iterative and incremental software development significantly increases development productivity (as compared to the traditional waterfall model).

6.2 Comments on Selected Influence Factors

In this section, we present an overview of comments on and experiences with selected factors influencing software development productivity.

6.2.1 Development team characteristics

Team size and structure is an important team-related factor influencing development productivity. In principle, the software engineering literature indicates significant benefits from using small teams. According to Camel and Bird [34], a common justification for small teams is not that a small team is so advantageous, but that a large team is disadvantageous. However, this does not seem to be consistent for the whole development life cycle. Brooks [29] says, for instance, that when we add people at the project's end, the productivity goes down. Balckburn et al. [22] support this thesis claiming that the small "tiger teams" in later stages of software development (after requirements specification) tend to increase productivity. Those experiences support the manpower Rayleigh buildup curve proposed in [97] and adapted in [106] where a larger team is used during early development phases and the team size drops during later phases.

However, the impact of team size on productivity has much to do with the communication and coordination structure [51][81][111]. In other words, the impact of team size depends significantly on the degree to which team members work together or separately, and on the communication structure [115] rather than simply on the number of people in a team. It has been observed that the main reason why large teams are usually so unproductive is primarily the burden of maintaining numerous communication links between the team members working together [34]. Thus, even large teams, where developers work mostly independently, do not seem to have a significant negative impact on productivity compared to small teams [115].

Even if large teams are indispensable, their negative influence on productivity can be moderated by an effective team (communication) structure and communication media. For instance, even in large-scale projects, minimizing the level of work concurrency by keeping several small, parallel teams (with some level of independence) instead of one large team working together seems to be a good strategy for preventing a drop in productivity [43]. That is, for example, why agile methods promoting two-person teams working independently (pair programming) may be (however only in certain conditions) more productive than large teams working concurrently [101]. Yet, as further reported in [101], well-defined development processes are not without significance in agile development. A large industrial case study proved, for example, that even pairs of experienced developers working together need a collaborative, role-based protocol to achieve productivity [101].

Observing social factors in hyper-productive organizations, Cain et al. [31] found the developer close to the core of the organization. Average and low productive organizations exhibit much higher prestige values for management functions than for those who directly add value to the product (developers). Collaborations in these organizations almost always flow from the manager in the center to other roles in the organization. Even though a *star structure* ("chief surgical team") is mostly the favored communica-

tion structure, it seems that its impact on productivity depends on who plays the central role, managerial (*outward control flow*) or technical (*inward control flow*) staff. The authors point to architect as an especially prestigious role in a team within highly productive organizations.

The use of proper *communication media* may also significantly improve team communication and, in consequence, development productivity. Face-to-face communication still seems to be the most efficient way of communication. Carey and Kacmar [33], for instance, observed that although simple tasks can be accomplished successfully using electronic communication media (e.g., telephone, email, etc.) complex tasks may result in lower productivity and larger dissatisfaction with electronic medium used. Agile software development considers one of the most effective ways for software developers to communicate to be standing around a whiteboard, talking, and sketching [40]. Ambler [5] goes further and proposes to define a specific period during the day (~5h) during which team members must be present together in the workroom. Outside that period, they may go back to their regular offices and work individually. A more radical option would be to gather the development team in a common workroom for the whole duration of the project. Teasley et al. [119], conclude their empirical investigation by saying that having development teams reside in their own large room (an arrangement called *radical collocation*) positively affected system development. The collocated projects had significantly higher productivity and shorter schedules than both the performance of similar past projects considered in the study and industry benchmarks. Yet, face-to-face communication does not guarantee higher productivity and may still vary widely dependent on the team communication structure (especially for larger teams).

Finally, it was observed that, besides a core team structure, close coupling to the quality assurance staff and the customer turned out to be a pattern within highly productive software organizations [31].

Staff turnover is another team-related project characteristic having a major impact on development productivity. Collofelo et al. [41] present an interesting study where, based on a process simulation experiment, they investigated the impact of various project strategies on team attrition. No action appeared to be the most effective strategy when schedule pressure is high and cost containment is a priority. Replacing a team member who left a team alleviates the team exhaustion rate (lightens increased attrition). Even though overstaffing is an expensive option, replacing a team member who left with several new members minimizes the duration of the project. Thus, this strategy should be considered for projects in which the completion date has been identified as the highest priority. It is, however, not clear how those results (especially regarding overstaffing) relate to Brooks' real-life observation that putting more staff at the end of a project makes it even more late [29].

Task assignment is the next reported human-related factor differentiating software development productivity. Boehm [24], for instance, introduces in his COCOMO II model the task assignment factor to reflect the observation that proper assignment of people to corresponding tasks has a great impact on development productivity. Hale et al. [59][115] go further and investigate related sub-factors such as *intensity*, *concurrency*, and *fragmentation*. They observed that the more time is spent on the task by the same developer (*intensity*), the higher the productivity. Further, the more team members work on the same task (*concurrency*), the lower the productivity (communication overhead disproportionately larger than task size). Finally, the more fragmented the developer's time over various tasks (*fragmentation*), the lower the productivity (due to overhead to switch work context).

Finally, ***team capabilities and skills*** is traditionally acknowledged as the most significant personnel characteristic that influences development productivity. Team capabilities also have an indirect impact on productivity by influencing other factors with a potential impact on development productivity. The impact of the team structure on productivity, for instance, usually assumes highly skilled technical peo-

ple. Moreover, the full benefits of tool support and technological improvements cannot be achieved without capable personnel [37]. On the other hand, the impact of team capabilities on productivity may be amplified or moderated by other factors. An appropriate management and communication structure can, for instance, moderate the negative impact of a low-experienced team on productivity; it can, however, never compensate for the lack of experience [22].

Researchers and practitioners agree about the high importance of team capabilities regarding development productivity. Yet, it is not clear exactly which skills have an impact on productivity, and which do not. Presented in Sections 4 and 5 suggest that programming language skills, domain experience, and project management skills are essential characteristics of a highly productive development team. Krishnan [81], however, claims that whereas higher levels of domain experience of the software team is associated with a reduction in the number of field defects in the product (higher quality), there is no significant direct association between either the language or the domain experience of the software team and the dollar costs incurred in developing the product. This does not, however, necessary imply no impact on productivity. Software teams with low experience (<2 years) may cost more due to lower productivity, whereas highly experienced teams (>10 years) may cost more due to higher salaries. Here we can again see that when observing development productivity and its influence factors, a careful definition of respective measures is crucial for the validity of the conclusions drawn.

Although not reflected by the results presented in this chapter (Sections 4 and 5), significant team capabilities do not only cover technical skills. There are several non-technical skills that are found to be essential with respect to development productivity. Examples are ability and willingness to learn/teach quickly, multi-area (general) specialization, knowledge of fundamentals, and flexibility [21]. It has been observed (especially in a packaged software development environment) that it is the innovation, creativity, and independence of the developers that determine development productivity and project success [35][59]. Yet, as warned in [9], an emphasis on creativity and independence creates an atmosphere that contributes to a general reluctance to implement accurate effort and time reporting. This, in turn, limits the reliability of the collected productivity data and may lead to wrong conclusions regarding productivity and its impact factors.

6.2.2 Schedule/effort distribution

The right schedule and work distribution are the next significant parameters influencing development productivity. Looking at the published investigations, one may get the impression that the development schedule should be neither too long nor to short. Thus both schedule over- and underestimation may have a negative impact on productivity.

Several authors have observed the negative impact of schedule compression on productivity (so-called *deadline effect*) [103][24]. A schedule compression of 25% (which is considered as *very low* compression) may, for instance, lead to a 43% increase in development costs [127]. Schedule compression is recognized as a key element by practically all of the most popular commercial cost estimation tools (e.g., PRICE-S [99], SLIM [106], SEER-SEM [69], COCOMO I/II [23][24]), which implement various productivity penalties related to it [127].

Others show that schedule expansion also seems to have a negative impact on development productivity [127][22][86]. One of the possible explanations of this phenomenon is the so-called Parkinson's low, which says that "the cost of the project will expand to consume all available resources" [100].

Yet, the right total schedule is only one part of success. The other part is its right distribution across single development phases. An investigation on productivity shown in [22] concludes that some parts of the process simply need to be "unproductive", i.e., should take more time. This empirical insight confirms

the common software engineering wisdom that more effort (and time) spent on early development stages bears fruit, for example, through less rework in later stages and finally increases overall project productivity. Apparently, the initial higher analysis costs of highly-productive projects are more than compensated for by the overall shorter time spent on the total project [84]. If the analysis phase, for example, is not done properly in the first place, it takes longer for programmers to add, modify, or delete codes. The Norden's Rayleigh curve [97] may here be taken as a reference shape of the most effective effort and time distribution over the software development life cycle. Yet, the simple rule of "the more the better" does not apply here and after exceeding a certain limit of time spent on requirements specification, the overall development productivity (and product quality) may drop [72]. The results of a case study at Ericsson [44] determined, for example, that the implementation phase had the largest improvement potential in the two studied projects, since it caused a large faults-slip-through to later phases, i.e., 85% and 86% of the total improvement potential of each project.

6.2.3 Use of tools and methods

The effect of the usage of certain tools and technologies on software development productivity is widely acknowledged (see Sections 4 and 5), but at the same time, paradoxically not well understood [74]. The opinions regarding the impact of tool usage vary largely across individual studies. Following the factors in Boehm's COCOMO II model [24], numerous organizations consider tool usage as a significant means for improving development productivity. Yet, empirical studies have reported their positive and negative effects [22] as well as little (no significant) direct effect [111]. This paradox might be explained by findings that the effect of tool usage on development productivity is coupled with the impact of other factors and that isolated use of tools makes little difference in the productivity results [1][21][20].

It has been observed, for instance, that due to the significant costs involved in learning the CASE tools, the effect of tools in some projects is negative, i.e., they increase the effort required for developing software products [58]. Other significant factors affecting the relationship between tool usage and development productivity are the degree of tool integration, tool maturity, tool training and experience, appropriate support when selecting and introducing corresponding tools (task-technology fit), team coordination/interaction, structured methods use, documentation quality, and project size [13][20][42][58][117]. For instance, when teams receive both tool-specific operational training and more general software development training, higher productivity is observed [58]. The same study reports on a 50% increase in productivity if tools and formal structured methods are used together.

Abdel-Hamid [1] observed, moreover, that studies conducted in laboratory settings using relatively small pilot projects tend to report productivity improvements of 100%- 600%, whereas when the impact of CASE tools is assessed in real organizational settings, much more modest productivity gains (15%-30% percent) or no gains at all were found. The author relevantly summarizes that project managers are the main source of failed CASE tools application, because most often they do not institute rudimentary management practices needed before and during the introduction of new tools. What they usually do is to look for a solution they can buy. Actually, in a simulation presented by Abdel-Hamid, almost half the productivity gains from new software development tools were squandered by bad management. Moreover, Ambler [5] underlines being flexible regarding tool usage. He observed that each development team works on different things, and each individual has different ways of working. Forcing inappropriate tools on people will not only hamper progress, it can destroy team morale.

The impact of project size and process maturity on the benefits gained from introducing tools is shown in [30]. The authors observed at IBM software solutions that in the context of advanced processes, productivity gains through requirements planning tools may vary between 107.9% for a small project (5

features) and –23.5% for a large project (80 features). Yet, in the context of a small project (10 features) such a gain may vary between -26.1% and 56.4% when regular and advanced processes are applied respectively. A replicated study confirmed this trend - productivity decreases with growing project size and higher process complexity. The productivity loss in the larger project was due to additional overhead for processing and maintaining a larger amount of data produced by a newly introduced tool. Higher process complexity, on the other hand, brings more overhead related to newly introduced activities (processes). Yet, measuring at the macro level makes it difficult to separate the impact of the tool from other confounding variables (such as team experience and size/complexity of a single feature). Therefore, the results of Bruckhaus [30] should be interpreted cautiously [10].

The use of software tools to improve project productivity is usually interpreted in terms of automated tools that assist with project planning, tracking, and management. Yet, non-automated tools such as checklists, templates, or guidelines that help software engineers interpret and comply with development processes can be considered as supporting the positive impact of high-maturity processes on improved project productivity [21].

6.2.4 Software reuse

Reuse of software products, processes, and other software artifacts is considered the technological key to enabling the software industry to achieve increased levels of productivity and quality [18][8].

Reuse contributes to an increase in productivity in both new development and software maintenance [14]. There are two major sources of savings generated by reuse: (1) less constructive work for developing software (reuse of similar and/or generic components) and (2) less analytical work for testing software and rework to correct potential defects (reuse of high-quality components). In other words, software reuse catalyzes improvements in productivity by avoiding redevelopment and improvements in quality by incorporating components whose reliability has already been established [112]. That is, for example, why reuse benefits are greater for more complex reusable components [94] (however, this positive effect levels off beyond a certain component size).

Nevertheless, reuse is not for free and may, at least at the beginning, bring negative savings (productivity loss). A high-quality, reusable (generic) software component first needs to be developed, which usually costs more that developing specific software (not intended to be reused). Populating the repository of reusable artifacts alone contributes to an initial loss of productivity [94]. The repository must not expand indefinitely, due to additional maintenance costs. On the one hand, creating and maintaining rarely used, small repositories of small components tends to cost more than the reuse savings they generate. As the repository size increases, opportunities for reuse tend to increase, generating more development savings. On the other hand, maintaining and searching through very large repositories may again generate negative reuse savings.

Later on, in order to actually reuse a certain component, it has to be identified as available in the repository and retrieved. Then, the feasibility of the component to be reused in a specific context has to be determined (a component has to be analyzed and understood). Even if accepted, a component often cannot be reused as is, but has to be modified in order to integrate it into the new product. At the end, it has to be (re)tested in a new context. In each step of reuse, a number of factors influencing its productivity have to be considered. Finally, if the total cost for retrieving, evaluating, and integrating a component is less than the cost of writing it from scratch, it makes economic sense to reuse the component.

The success of reuse depends on numerous factors. Rinie and Sonnemann [109] used an industrial survey to identify several leading reuse success factors (so-called *reuse capability indicators*):

- Use of product-line approach,
- Architecture that standardizes interfaces and data formats,
- Common software architecture across the product line,
- Design for manufacturing approach,
- Domain engineering,
- Management that understands reuse issues,
- Software reuse advocate(s) in senior management,
- Use of state-of-the-art tools and methods,
- Precedence of reusing high-level software artifacts such as requirements and design versus just code reuse, and
- Tracing end-user requirements to the components that support them.

Atkins et al. [10][11] confirm part of these results in an empirical study where the change effort of large telecommunication software was reduced by about 40% by using a version-sensitive code editor and about four times when the domain engineering technologies were applied.

The impact of reuse on development productivity, like most other influence factors, is strongly coupled with other project characteristics. It should thus not be simplified by taking as granted the positive impact of reuse on productivity. Frakes and Succi [52] observed, for instance, a certain inconsistency regarding the relationship between reuse and productivity across various industrial data sets, with some relationships being positive and others negative.

One of the factors closely coupled with reuse are the characteristics of the personnel involved in reuse (development of reusable assets and their reuse). Morisio et al. [93] observed that the more familiar developers are with reused, generic software (framework), the more benefit is gained when reusing it. The authors report that although developing a reusable asset may cost 10 times as much as "traditional" development, the observed productivity gain of each development where the asset is reused reached 280%. Thus, the benefit from creating and maintaining reusable (generic) assets increases with the number of its reuses.

Another factor influencing the impact of reuse on development productivity is the existence of defined processes. Soliman [116], for example, identifies the lack of a strategic plan available to managers to implement software reuse as a major factor affecting the extent of productivity benefits gained from reuse. Major issues for managers to consider include: commitments from top management, training for software development teams, reward systems to encourage reuse, measurement tools for measuring the degree of success of the reuse program, necessary reuse libraries, rules regarding the development of reusable codes, domain analysis, and an efficient feedback system.

Software reuse in the context of **object-oriented (OO) software development** is one specific type of reuse that proved to be especially effective in increasing development productivity [15][83]. Yet, the common belief that using the OO paradigm is sufficient for improving development productivity does not have much quantitative evidence supporting it [83]. OO reuse requires considering several aspects, such as the type of reuse or the domain context of reuse. Results of experiments in the context of C++ implementation show, for instance, that since *black-box reuse* (class reuse without modification) increases programmer productivity, the benefits of *white-box reuse* (reuse by deriving a new class from an existing one through inheritance) are not so clear (especially for reuse of third-party library classes) [82].

Finally, introducing the OO paradigm, like introducing any new technology, requires considering and often adjusting the business context (model) in order to gain the expected productivity benefits [103]. For example, inappropriate schedule planning (constrained or too long) or lack of process control (e.g., over the effects of newly introduced technology) may completely level down any benefit from the applied new technology, including the object-oriented paradigm [103].

The use of **commercial-off-the-shelf (COTS)** components is closely related to software reuse and might, in practice be classified as a subclass of software reuse. Yet, software reuse differs from COTS software in three significant ways [1]: (1) reuse components are not necessarily able to operate as standalone entities (as is assumed to be the case with most components defined as COTS software); (2) reuse software is generally acquired internally within the software developing organization (by definition, COTS components must come from outside); and (3) reused software usually requires access to the source code, whereas with COTS components, access to the source code is rare (references to so-called "white box" COTS notwithstanding). From that perspective, the productivity of software development in the context of software reuse and use of COTS is influenced by overlapping sets of factors.

In principle, the factors presented in the literature focus on the productivity of integrating COTS into developed software [1]. Abts [2] postulates, for instance, that increasing the number of COTS components is economically beneficial only up to a certain point, where the savings resulting from the use of COTS components break even with the maintenance costs arising from the volatility of such components.

Finally, **software product lines** and **domain engineering** as the most recent incarnations of the reuse paradigm should be considered as a potential factor influencing productivity of software development in general, and software changes in particular. In fact, they may potentially reduce the cost of software change three to four-fold (telecommunication switch system) [91][114]. Such a gain is in line with the generally accepted view that domain engineering techniques improve software productivity two to ten-fold. Yet, the cost of change is also influenced by other factors. Besides, quite obviously, the size and complexity of change, the type of change also has a substantial impact on change productivity. Similarly to [55], the authors observed, for instance, that changes that fix bugs are more difficult than comparably sized additions of new code (see also Section 6.1.3).

Concluding, there are various aspects of software reuse (besides simply the level of reuse) that have to be taken into account when considering reuse as a way to increase development productivity in a certain context. In practice, it is recommended to collect organization-specific data related to reuse and productivity, and use these data to guide reuse projects. However, one has to be careful when defining both productivity and reuse measures for that purpose, since improper measure definition might lead to wrong conclusions [46].

6.2.5 Software outsourcing

Outsourcing has recently been gaining a lot of attention from the software community as a means for reducing development costs. According to SPR data [71], the level of outsourcing tripled in the years 1989-2000. In 2000, about 15% of all information systems were produced under contract or outsource agreements (2% international outsource).

Now, it is not clear if reduced cost is related to lower man-power costs or higher development productivity at organizations providing outsourcing services. Analyzing the ISBSG data [68], Cabonneau [32] confirmed the results of earlier studies (e.g., [49]) that outsourced projects do not have significantly different productivity (in terms of functionality delivered per effort invested) than in-source projects. Therefore software development outsourcing will only lead to significant cost savings when the out-

sourcing provider has access to significantly cheaper labor. This is consistent with prior research [126], which concludes that "an external developer must have a considerable cost advantage over an internal developer in order to have the larger net value".

Nowadays, companies are more and more interested not only in lower labor cost but also in increased productivity of outsourcing projects. A study performed across 15 business units of a large international software organization performed by Fraunhofer IESE showed that 42% of the respondents plan to use productivity measurement to actually manage outsourcing projects. For that purpose, they need to identify factors that influence productivity in the context of outsourcing.

In case of outsourcing projects, communication between software provider and outsourcing organization seems to be a crucial aspect influencing development productivity. As already mentioned in Section 6.2.1, the number of involved people (team size) as well as communication structure have a significant impact on development productivity. This is especially true for outsourced projects [62][63], since outsourcing projects usually suffer from a geographical and mental distance between the involved parties. Software might be outsourced to an organization in the same country (near-shore) or abroad (far-shore). Due to geographical, temporal, and cultural distances, international outsourcing is found to be between 13 and 38% more expensive than national outsourcing [4]. In that context, communication facilities play an essential role. A summary of communication means in the context of off-shoring projects can be found, for instance, in Moczadlo [92] (Figure 6-2).

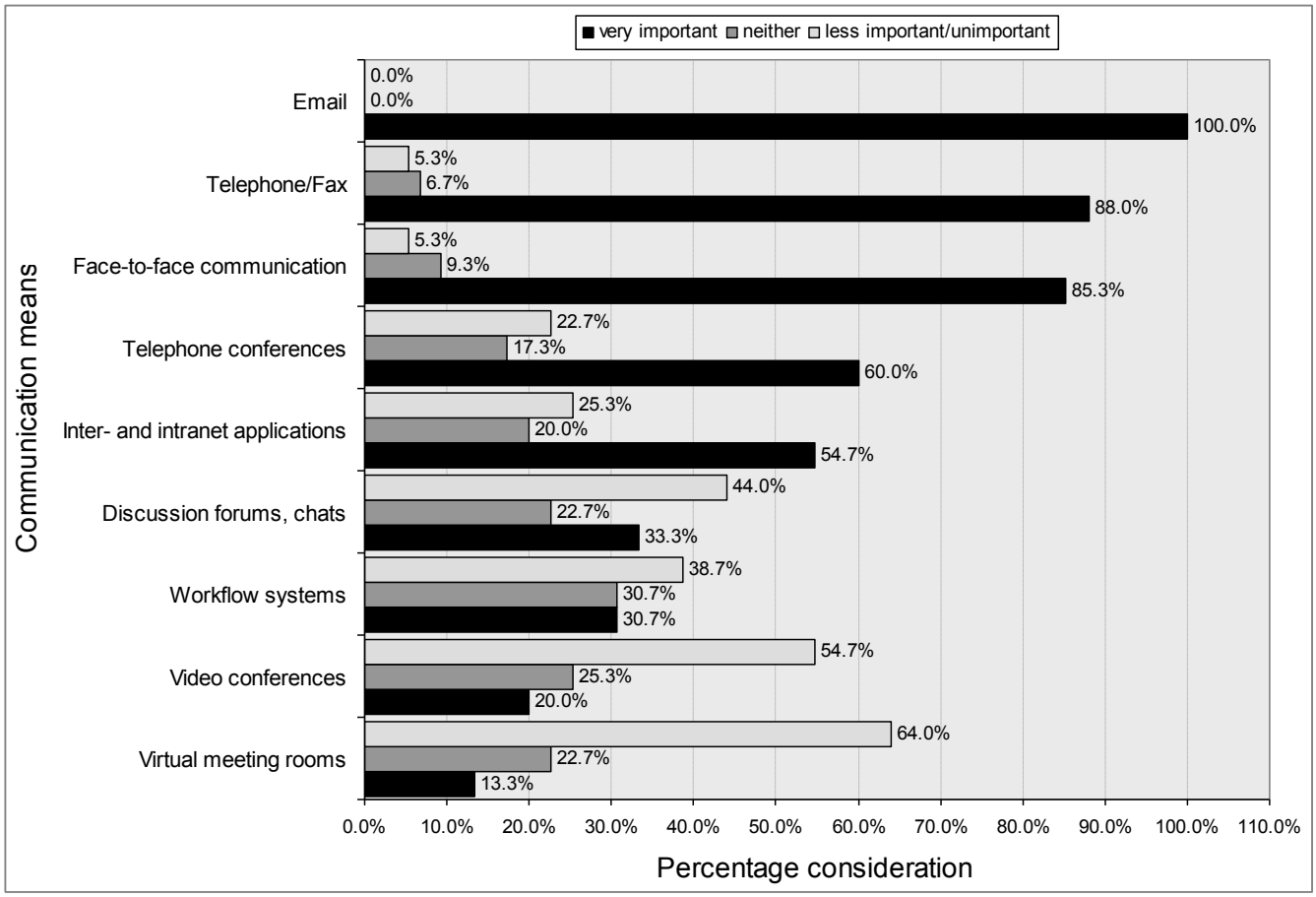

Figure 6-2 Importance of communication channels [92]

Team and task distribution is another project aspect related to communication between project stakeholders (see also section 6.2.1) and thus should be considered in the context of software outsourcing. DeNeve and Ebert [45] strongly advise building coherent and collocated teams of fully allocated engineers. Coherence means splitting the work during development according to feature content and assembling a team that can implement a set of related functionality. Collocation means that engineers working on such a set of coherent functionality should sit in the same building, perhaps within the same room. Full allocation implies that engineers working on a project should not be distracted by different tasks in other projects.

Another already known factor that influences the productivity of outsourcing projects is *project management*, including realistic, detailed, and well-structured project planning [45][61]. Those aspects cover factors such as schedule pressure and distribution (see Section 6.2.2), as well as task assignment (see Section 6.2.1).

Customer/contractor involvement, which is less important in case of in-house development (see results in Sections 4 and 5), becomes essential in outsourcing, where a software product is developed completely by a service provider [61]. It is recommended, for instance, that "at least one technical staff from the contracting organization should be involved in the details of the work, to form a core of technical expertise for in-house maintenance" [61].

7. CONSIDERING PRODUCTIVITY FACTORS IN PRACTICE

There are several essential issues that must be considered when selecting factors for the purpose of modeling software development productivity, effort/cost, schedule, etc.

This section presents a brief overview of the most important aspects to be considered in practice.

7.1 Factor Definition & Interpretation

Despite similar naming, the productivity factors presented in the literature usually differ significantly across various studies with respect to the phenomenon they actually represent. Moreover, our experience (e.g., [122]) is that, in practice, even if factor definitions suggest similar underlying phenomena, these may be interpreted differently by various experts - dependent on their background and experiences. Software reliability, for instance, is sometimes defined (or at least tacitly interpreted) as including safety and security and sometimes not.

Furthermore, a high-level view on abstract productivity factors that aggregate a number of specific indicators within a single factor may mask potential productivity problems, as poor results in one area (indicator) can be offset by excellence in another [9]. Therefore, it is recommended identifying project aspects influencing productivity on the level of granularity ensuring identification of potentially offsetting factors. Team capabilities, for instance, may cover numerous specific skills such as communication or management skills. It may thus be that low communication skills of developers are not visible because they are compensated by great management skill of the project manager. In such a case, considering team capabilities as a productivity indicator would not bring much value.

Therefore, in order to maximize the validity of the collected inputs (factor's significance, factor's value, and factor's impact on productivity), the exact definition of the factors as well as related measures has to be done in the first place. Blindly adopting published factors and assuming that everyone understands them may consistently (and usually does) lead to large disappointments. Unclear factor definition results

in invalid data and inaccurate and instable models. In consequence, after investing significant effort, software organizations finally completely give up quantitative project management.

7.2 Factor Selection

One of the major purposes of the overview presented in this chapter is to support software practitioners in selecting factors relevant in their specific context. We are, however, far from suggesting uncritical adoption of the "top" factors presented here. This chapter is supposed to increase a software practitioner's understanding of possible sources of productivity drivers rather than bias his thinking while selecting factors relevant in his specific context. The overview presented here should, at most, be taken as a starting point and reference for selecting context-specific factors.

As with any decision support technology, the prerequisite when selecting productivity factors is that they maximize potential benefit while minimizing related costs. On the business level, they should then contribute to the achievement of organizational goals, such as productivity control and improvement, while generating minimal additional costs. On the software project level, the selected factors should cover (explain) a possibly large part of the observed productivity variance and be of minimal quantity in order to assure an acceptable cost of collecting, interpreting, and maintaining respective project data.

Therefore, an effective factor selection approach is an essential step in modeling productivity or any productivity-related phenomena.

Selecting an optimal[6] set of productivity factors is not a trivial task. In principle, we distinguish three major selection approaches: based on experts' assessments, based on data analysis, and a hybrid approach where both data- and expert-based methods are combined. Industrial experiences (e.g., [122]) indicate that none of the first two methods is able to provide us with an optimal set of factors.

Data-based factor selection techniques provide a subset of already measured factors that has usually been selected arbitrarily, e.g., based on one of the popular cost estimation models [36][77]. Data-based selection can usually be easily automated and thus does not cost much (in terms of manpower[7]). One significant limitation is that data-based selection simply reduces the set of factors given a priori as input. This means in practice that if input data does not cover certain relevant productivity factors, a data-based approach will not identify them. It may at most exclude irrelevant factors. Maximizing the probability of covering all relevant factors would require collecting a significant amount of data, hopefully covering all relevant factors, which would most probably disqualify such an approach due to high data collection costs.

On the other hand, *expert-based factor selection techniques* seem to be more robust, since experts are able to identify (based on their experience) factors unmeasured so far. However, experts tend to be very inconsistent in their assessments, depending, for example, on personal knowledge and expertise. Across 17 IESE studies where we asked experts to rank identified factors with respect to their impact on productivity and where we measured Kendall's coefficient of concordance $W \in (0, 1)$ [113] to quantify experts' agreement, in half of the cases (46%) experts disagreed significantly ($W <= 0.3$ at a significance level $p=0.05$).

[6] A minimal amount of factors that would meet specified cost- and benefit-related criteria, e.g., effective productivity control at minimal modeling cost.

[7] Factor selection techniques are easy to automate; however, many of them contain NP-hard algorithms (e.g., [12]), which actually limits their practical applicability.

Hybrid approaches to selecting productivity factors seems to be the best alternative. In the reviewed literature, however, merely 6% of the studies directly propose some kind of combined selection approach. Most of the published studies (45%) select productivity factors based on experts' opinion or already published factors (with COCOMO factors [24] coming out on top). A successful attempt at combining data- and expert-based approaches within an iterative framework has been made, for example, in [122].

7.3 Factor Dependencies

In practice, productivity factors are not independent of each another. Identification of reciprocal relationships between productivity factors is a crucial aspect of productivity modeling. On the one hand, explicit consideration of factors' dependencies provides software practitioners with a more comprehensive basis for decision making. On the other hand, the existence of relationships between productivity factors limits the applicability of certain modeling methods (e.g., multiple regression models) and thus must be known in advance, before applying certain modeling methods.

In practice, we may distinguish between several kinds of between-factor relationships. Factors may be in a causal relationship, which means that a factor's change (*cause*) leads to (*causes*) a change (*effect*) to a related factor. A factor may also be correlated, which means that changes to a factor go in parallel with changes to another factor. Although factors linked in a causal relationship should be correlated, correlation does not imply causal association.

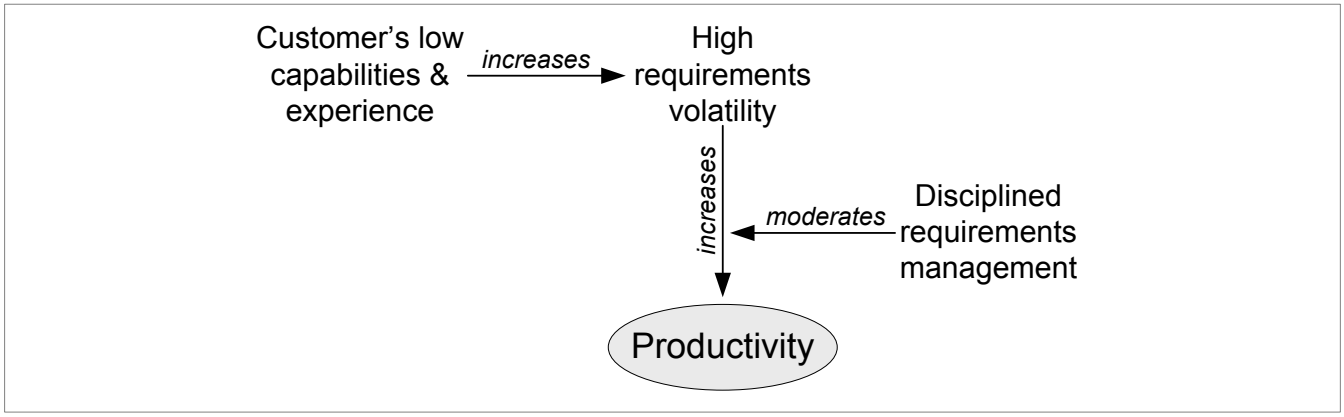

Figure 7-1 Types of factor relationships

Moreover, besides influencing another factor, a factor may also influence the strength of another factor's impact on productivity. For example (Figure 7-1), *Customer's low capabilities & experiences* contributes to *Higher requirements volatility*, which in turn leads to higher development effort. Now, the negative impact of *Higher requirements volatility* can be moderated (decreased) by *Disciplined requirements management*.

There are several practical issues to be considered regarding factor dependencies. The first one is how to identify various kinds of dependencies? Data-based (analytical) methods are able at most to identify correlations. Cause-effect relationships can be identified by experts, who, however, tend to disagree in their subjective assessments. The next issue is what should we do with the identified dependencies? Should we explicitly model or better eliminate them? What alternative techniques exist to model or eliminate the identified relationships? Existing publications on productivity measurement/modeling do not explicitly pay much attention to those issues.

7.4 Model Quantification

Quantitative productivity management requires systematic collection of related project data. This entails quantifying selected productivity factors and their impact on productivity.

Proper definition of measures for each identified factor later on contributes to the cost and reliability of collecting the data upon which quantitative productivity management will take place. Non-continuous factors (nominal, ordinal), especially, require a clear and unambiguous definition of scales, reflecting distinct (orthogonal) classes. Wrong measurement processes and improper scales definition usually lead to unreliable, messy data, which significantly limits the applicability of certain analysis methods and the reliability of the obtained analysis results (see, e.g., [122]).

Finally, the impact each identified factor has on productivity and, if explicitly modeled, on other factors should be quantified. Some statistical approaches, e.g., regression, provide certain weights that reflect each factor's impact on productivity. Data mining provides a set of techniques dedicated to weight a factor's impact (e.g., [12]). One major disadvantage of those methods is that the weights they provide are rather hard to interpret by experts. There are also several approaches based on expert assessments to quantify a factor's impact on productivity. The CoBRA method [122], for instance, uses a percentage change of productivity caused by the worst-case factor value, which is very intuitive and easy to asses by experts. A less intuitive measure is used within Bayesian Belief Nets [50], where the conditional probability of a certain productivity value given the values of the influencing factors is assessed.

8. SUMMARY AND CONCLUSIONS

This chapter has presented a comprehensive overview of the literature and of experiences made within Fraunhofer IESE regarding the most common factors influencing software development productivity.

The major outcome of the study is that the success of software projects still relies upon humans.

The second most commonly considered factors are tool and method. However, even the best tool or method alone is not a silver bullet and cannot be a substitute for highly skilled people and effective work coordination. Investing in people is still considered as bringing more benefit than investing in tools and methods only [22]. Tools and methods should therefore be considered as human aid that amplifies the positive impact of highly-skilled and well-coordinated teams on development productivity [111].

Some productivity factors refer to using or not using certain activities. Yet, a certain activity may be consistently applied across development projects and therefore, at first glance, not by considered as having an impact on development productivity. However, when we consider the effectiveness of a certain activity, it may occur that it still has a significant impact on productivity. This calls for considering software development methods, processes, and tools in terms of their effectiveness rather than simply using or not using them.

Moreover, any software development effort, even if staffed with skilled individuals, is likely to be unsuccessful if it does not explicitly account for how people work together [111]. A software development environment is a complex social system that may squander the positive impact of skillful individuals as well as software tools and methods if team communication and coordination fail [1].

Factors facilitating team communication and work coordination are particularly important in the context of software outsourcing. Geographical and, often, mental distance between the involved parties (e.g., outsourcing company, software provider, etc.) require dedicated managerial activities and communication facilities to maintain a satisfactory level of productivity.

The most commonly selected factors support the thesis that schedule is not a simple derivative of project effort. The negative impact of project schedule on productivity, however, is considered only in terms of schedule constraints (compression). Parkinson's low ("cost of the project will expand to consume all available resources") seems not to be considered in daily practice.

Several "top" factors support the common intuition regarding the requirements specification as the key development phase. First of all, requirements quality and volatility are considered to be essential drivers of development productivity. Several further factors are considered as either contributing to the quality and volatility of requirements or moderating the impact of already instable requirements on productivity. Distribution of the project effort (manpower) focusing on the requirements phase as well as significant customer involvement in the early phases of the development process are the factors most commonly believed to increase requirements quality and stability. The impact of already instable requirements may, on the other hand, be moderated by disciplined requirements management as well as early reviews and inspections.

Finally, the results obtained here do not support the traditional belief that software reuse is the key to productivity improvements. It seems that the first years of enthusiasm also brought much disappointment. A plethora of factors that should be considered in order to gain the expected benefits from reuse might explain this situation. Ad hoc reuse, without any reasonable cost-benefit analysis and proper investments to create a reuse environment (e.g., creation and maintenance of high-quality reusable assets, integration support, appropriate team motivation and training) usually contributes to a loss in productivity.

The factors presented in this chapter result from a specific aggregation approach that reflects current industrial trends. However, it must be considered that the analyzed studies usually differ widely with respect to the identified factors, their interdependencies, and their impact on productivity. Therefore, each organization should consider potential productivity factors in its own environment ("what is good for them does not have to necessarily be good for me"), instead of uncritically adopting factors used in other contexts (e.g., COCOMO factors) [5]. Moreover, since software is known as a very rapidly changing environment, selected factors should be reviewed and updated regularly.

Selecting the right factors is just a first step towards quantitative productivity management. The respective project data must be collected, analyzed, and interpreted from the perspective of the stated productivity objectives [17][19]. Inconsistent measurements and/or inadequate analysis methods may, and usually do, lead to deceptive conclusions about productivity and its influencing factors [46]. In that sense, one may say that rigorous measurement processes and adequate analysis methods also have a significant impact on productivity, although not directly [26]. Therefore, such aspects as clear definition and quantification of selected factors, identification of factor interdependencies, as well as quantification of their impact on productivity has to be considered.

ACKNOWLEDGEMENTS

We would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of this chapter.

REFERENCES

[1] Abdel-Hamid, T.K., "The Slippery Path to Productivity Improvement," *IEEE Software*, 13(4):43-52, July 1996.

[2] Abts, C.M., Boehm B.W., *COTS Software Integration Cost Modeling Study*. University of Southern California, Center for Software Engineering, June 1997.

[3] Albrecht, A.J., ”Measuring Application Development Productivity,” *Proceedings of the IBM Applications Development Symposium*, Monterey, California, October 14-17, 1979, pp. 83-92.

[4] Amberg, M., Wiener, M. *Wirtschaftliche Aspekte des IT Offshoring* [Economic Aspects of IT Offshoring], Arbeitspapier. 6, Universität Erlangen-Nürnberg, Germany, 2004 (in German).

[5] Ambler, S.M., “Doomed from the start: what everyone but senior management seems to know,” *Cutter IT Journal*, 17(3):29-33, 2004.

[6] Andersson, C., Karlsson, L., Nedstam, J., Höst, M., Nilsson, B., “Understanding Software Processes through System Dynamics Simulation: A Case Study,” *Proceedings of the 9th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 8-11 April 2002, pp. 41-50.

[7] Angkasaputra, N., Bella, F., Hartkopf, S., *Software Productivity Measurement - Shared Experience from Software-Intensive System Engineering Organizations*. IESE-Report No. 039.05/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2005.

[8] Angkasaputra, N., Bella, F., Berger, J., Hartkopf, S., Schlichting A., *Zusammenfassung des 2. Workshops „Software-Produktivitätsmessungen“ zum Thema Produktivitätsmessung und Wiederverwendung von Software* [Summary of the 2nd Workshop “Software Productivity Measurement“ on Productivity Measurement and Reuse of Software], IESE-Report Nr. 107.05/D, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2005 (in German).

[9] Anonymous, *Above average(s): measuring application development performance*. Intranet and Networking Strategies Report, 8(3):1-4, 2000.

[10] Atkins, D.L., Ball, T., Graves, T.L., Mockus, A., “Using version control data to evaluate the impact of software tools: a case study of the Version Editor,“ *IEEE Transactions on Software Engineering*, 28(7):625-637, July 2002.

[11] Atkins, D.L., Mockus, A., Siy, H.P., “Measuring technology effects on software change cost,“ *Bell Labs Technical Journal*, 5(2):7-18, April/June 2000.

[12] Auer, M., Trendowicz, A., Graser, B., Haunschmid, E., Biffl, S., “Optimal Project Feature Weights in Analogy-Based Cost Estimation: Improvement and Limitations,“ *IEEE Transactions on Software Engineering*, 32(2):83-92, 2006.

[13] Baik, J., Boehm, B.W., Steece, B.M., “Disaggregating and Calibrating the CASE Tool Variable in COCOMO II,“ *IEEE Transactions on Software Engineering*, 28(11):1009-1022, November 2002.

[14] Basili, V., “Viewing maintenance as reuse-oriented software development,” *IEEE Software,* 7(1):19-25, January 1990.

[15] Basili, V.R., Briand, L.C., Melo, W.L., “How Reuse Influences Productivity in Object-Oriented Systems,“ *Communications of the ACM*, 39(10):104-116, 1996.

[16] Basili, V., Briand, L., Condon, S., Kim, Y-M., Melo, W.L., Valen, J.D., “Understanding and predicting the process of software maintenance releases,“ *Proceeding of the 18th International Conference on Software Engineering*, Berlin, Germany, pp. 464-474, 1996.

[17] Basili, V.R., *Software Modeling and Measurement: The Goal Question Metric Paradigm*. Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, USA, September 1992.

[18] Basili, V., Rombach, H.D., “The TAME project: Towards improvement-oriented software environments,” *IEEE Transactions on Software Engineering*, 14(6): 758-773, June 1988.

[19] Basili, R., Weiss, D.M., "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, SE-10(6):728-737, November 1984.

[20] Bazelmans, R., "Productivity - The role of the Tools Group," *ACM SIGSOFT Engineering Notes*, 10(2):63-75, July 1985.

[21] Bechtold, R., "Reducing Software Project Productivity Risk," *CrossTalk - The Journal of Defense Software Engineering*, 13(5):19-22, May 2000.

[22] Blackburn, J.D., Scudder, G.D., Van Wassenhove, L., "Concurrent software development," *Communications of the ACM*, 43(4):200-214, 2000.

[23] Boehm, B.W., *Software Engineering Economics*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[24] Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Refer, D., Steece, B., *Software Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[25] Boehm, B.W., Sullivan, K.J., "Software economics: a roadmap," *Proceedings to the International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 319-343.

[26] Hai, Suan-Bok, Raman, K.S., "Software engineering productivity measurement using function points: a case study," *Journal of Information Technology Cases and Applications*, 15(1):79-90, 2000.

[27] Briand, L.C., El Emam, K., Bomarius, F., "COBRA: A Hybrid Method for Software Cost Estimation, Benchmarking and Risk Assessment," *Proceedings of the 20th International Conference on Software Engineering*, pp. 390-399, April 1998.

[28] Briand, L.C., Wieczorek, I., "Software Resource Estimation," in: Marciniak J.J. (ed.), *Encyclopedia of Software Engineering*, John Wiley & Sons, 2:1160-1196, 2002.

[29] Brooks, F.P. *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition. Reading, MA: Addison-Wesley, 1995.

[30] Bruckhaus, T., Madhavii, N.H., Janssen, I., Henshaw, J., "The impact of tools on software productivity," *IEEE Software*, 13(5):29-38, September 1996.

[31] Cain, B.G., Coplien, J.O., Harrison, N.B., "Social patterns in productive software development organizations," *Annals of Software Engineering*, 2(1):259-286, 1996.

[32] Carbonneau, R., *Outsourced Software Development Productivity*. Report MSCA 693T. John Molson School of Business, Concordia University. 2004.

[33] Carey, J.M., Kacmar, C.J., "The impact of communication mode and task complexity on small group performance and member satisfaction," *Computer in Human Behaviour*, 13(1):23-49, 1997.

[34] Carmel, E., Bird, B.J., "Small is beautiful: A study of packaged software development teams," *Journal of High Technology Management Research*, 8(1):129-148, 1997.

[35] Carmel, E., Sawyer, S., "Packaged Software Teams: What Makes Them So Special?" *Information Technology and People*, 11(1):6-17, 1998.

[36] Chen, Z., Menzies, T., Port, D., Boehm, B., "Finding the Right Data for Software Cost Modeling," *IEEE Software*, 22(6):38-46, November/December 2005.

[37] Chiang, I.R., Mookerjee, V.S., "Improving Software Team Productivity," *Communications of the ACM*, 47(5):89-93, May 2004.

[38] CMMI Project Team, *CMMISM for Software Engineering, Version 1.1, Staged Representation*. Technical Report CMU/SEI-2002-TR-029, Carnegie Mellon Software Engineering Institute, Pittsburg, August 2002.

[39] Clark B.K., "Quantifying the Effects of Process Improvement on Effort," *IEEE Software*, 17(6):65-70, November/December 2000.

[40] Cockburn, A., *Agile Software Development*. Addison-Wesley Professional, Boston, USA, 2001.

[41] Collofello, J., Houston, D., Rus, I., Chauhan, A., Sycamore, D.M., Smith-Daniels, D., "A system dynamics software process simulator for staffing policies decision support," *Proceedings to the 31st Annual Hawaii International Conference on System Sciences*, 6-9 January1998, Kohala Coast, HI, USA, 6:103-111.

[42] Cruz, C.D., "A proposal of an object oriented development cost model," *Proceedings of the European Software Measurement Conference*, Antwerp, Belgium, Technologisch Instituut VZW,1998, pp. 581-587.

[43] Cusumano, M., Selby, R., "How Microsoft Builds Software," *Communications of the ACM*, 40(6):53-61, 1997.

[44] Damm, L.O., Lundberg, L. and Wohlin, C., "Faults-slip-through - a concept for measuring the efficiency of the test process," *Software Process Improvement and Practice*, 11(1):47-59, January/February 2006.

[45] de Neve, P., Ebert, C., "Surviving Global Software Development," *IEEE Software*, 18(2):62-69, 2001.

[46] Devanbu, P., Karstu, S., Melo, W., Thomas, W., "Analytical and empirical evaluation of software reuse metrics," *Proceedings of the 18th International Conference on Software Engineering*, p. 189, 1996.

[47] Desharnais, J.-M., *Analyse statistique de la productivite des projects de development en informatique apartir de la technique des points des function*. Master's Thesis, University of Montreal, Canada, 1989 (in French).

[48] Diaz, D., King, J., "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line," *CrossTalk - The Journal of Defense Software Engineering*, 15(3):9-14, March 2002.

[49] Earl, M.J. "The Risks of Outsourcing IT," *loan Management Review*, 37(3):26-32, Spring 1996.

[50] Fenton, N., Marsh, W., Neil, M., Cates, P., Forey, S., Tailor, M., "Making resource decisions for software projects," *Proceedings of the 26th International Conference on Software Engineering*, pp. 397- 406, May 2004.

[51] Fenton, N.E., Pfleeger, S.L., *Software Metrics. A Rigorous & Practical Approach*, 2nd edition, International Thomson Computer Press, London, UK, 1997.

[52] Frakes, W.B., Succi, G., "An industrial study of reuse, quality, and productivity," *The Journal of Systems and Software,* 57:99-106, 2001.

[53] Gartner Inc. press releases, *Gartner Survey of 1,300 CIOs Shows IT Budgets to Increase by 2.5 Percent in 2005*, 14th January 2005, http://www.gartner.com/press_releases/pr2005.html.

[54] Gartner Inc. press releases, *Gartner Says Worldwide IT Services Revenue Grew 6.7 Percent in 2004*, 8th February 2005, http://www.gartner.com/press_releases/pr2005.html.

[55] Graves, T.L., Mockus, A., "Inferring Change Effort from Configuration Management Databases," *Proceedings of the 5th International Software Metrics Symposium*, Bethesda, MD, USA, pp. 267-273, 1998.

[56] Greves, D., Schreiber, B., Maxwell, K., Van Wassenhove, L., Dutta, S., The ESA "Initiative for Software Productivity Benchmarking and Effort Estimation," *European Space Agency Bulletin*, (87), August 1996.

[57] Griffyth, J., "Human factors in high integrity software development: a field study," *Proceedings of the 15th International Conference on Computer Safety, Reliability and Security*, Vienna, Austria, 23-25 October 1997, London, UK, Springer Verlag, 1997.

[58] Guinan, P., Cooprider, J., Sawyer, S., "The Effective Use of Automated Application Development Tools," *IBM Systems Journal*, 36(1):124-139, 1997.

[59] Hale, J., Parrish, A., Dixon, B., Smith, R.K., "Enhancing the COCOMO estimation models," *IEEE Software*, 17(6):45-49, November/December 2000.

[60] Harter. D.E., Krishnan, M.S., Slaughter, S.A., "Effect of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science*, 46(4):451-466, April 2000.

[61] Herbsleb, J., Paulish, D., Bass, M., "Global Software Development at Siemens: Experience from Nine Projects," *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, Missouri, USA, 2005.

[62] Herbsleb, J. D., Mockus, A., "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Transactions on Software Engineering*, 29(6):481-494, 2003.

[63] Herbsleb, J. D., Mockus, A., Finholt, T. A., Grinter R. E., "An Empirical Study of Global Software Development: Distance and Speed," *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Canada, 2001.

[64] Herron, D., Garmus, D., "Identifying Your Organization's Best Practices," *CrossTalk - The Journal of Defense Software Engineering*, 18(6):22-25, June 2005.

[65] Heemstra, F.J., van Genuchten, M.J.I.M., Kusters, R.J., *Selection of Cost Estimation Packages*. Research report EUT/BDK/36, Eindhoven University of Technology, Eindhoven, Netherlands, 1989.

[66] Heidrich, J., Trendowicz, A., Münch, J., Wickenkamp, A., *Zusammenfassung des 1st International Workshop on Efficient Software Cost Estimation Approaches, WESoC'2006*. IESE Report 053/06E, Fraunhofer Institute for Experimental Software Engineering, April, 2006 (in German).

[67] *IEEE Std 1045-1992. IEEE Standard for Software Productivity Metrics*. IEEE Computer Society, 1992.

[68] *ISBSG Data Repository. Release 9*, International Software Benchmarking Group, Australia, 2005.

[69] Jensen, R.W., "An Improved Macrolevel Software Development Resource Estimation Model," *Proceedings of the 5th International Society of Parametric Analysts Conference*, St. Louis, Mo. April 26-28, 1983, pp. 88-92.

[70] Jones, C., "Software Cost Estimating Methods for Large Projects," *CrossTalk - The Journal of Defense Software Engineering*, 18(4):8-12, April 2005.

[71] Jones, C., *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Inc., New York, USA, 2000.

[72] Jones, T.C., *Estimating Software Cost*, McGraw-Hill, New York, USA, 1998.

[73] Jørgensen, M., Shepperd, M., "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Transactions on Software Engineering*, 33(1):33-53, January 2007.

[74] Kemerer, C.F. *Software Project Management Readings and Cases*. McGraw-Hill, Chicago, USA, 1997.

[75] Kemerer, C.F., "An empirical validation of software cost estimation models," *Communications of the ACM*, 30:416-429, May 1987.

[76] Kennedy, K., Koelbel, C., Schreiber, R., "Defining and measuring productivity of programming languages," *International Journal of High Performance Computing Applications*, 11(4):441-448, Winter 2004.

[77] Kirsopp, C., Shepperd, M.J., Hart, J., "Search Heuristics, Case-based Reasoning and Software Project Effort Prediction," *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. pp. 1367-1374.

[78] Kitchenham, B., *Procedures for Performing Systematic Reviews*, Technical Report TR/SE-0401, Keele University, Keele, UK, 2004.

[79] Kitchenham, B.A., Mendes, E., "Software Productivity Measurement Using Multiple Size Measures," *IEEE Transactions on Software Engineering*, 30(12):1023-1035, December 2004.

[80] Kitchenham, B.A., Taylor, N.R., "Software project development cost estimation," *Journal of Systems and Software*, 5:267-278, 1985.

[81] Krishnan, M.S. "The role of team factors in software cost and quality: an empirical analysis," *Information Technology and People*, 11(1):20-35, 1998.

[82] Lattanzi, M., Henry, S., "Software reuse using C++ classes. The question of inheritance," *The Journal of Systems and Software*, 41:127-132, 1998.

[83] Lewis, J. A.., Henry, S. M., Kafura, D. G., "An Empirical Study of the Object-Oriented Paradigm and Software Reuse," *Proceedings of the Conference on Object-oriented Programming Systems*, Languages and Applications, pp. 184-196, 1991.

[84] Mahmood, M.A., Pettingell, K.J., Shaskevich, A.I., "Measuring productivity of software projects: a data envelopment analysis approach," *Decision Sciences*, 27(1):57-80, 1996.

[85] Mair, C., Shepperd, M., Jorgensen, M., "An Analysis of Data Sets Used to Train and Validate Cost Prediction Systems," *Proceedings of International Workshop on Predictor Models in Software Engineering*, May 15, 2005, St. Louis, Missouri, USA., pp. 1-6.

[86] Maxwell, K.D., Van Wassenhove, L., Dutta, S., "Software Development Productivity of European Space, Military, and Industrial Applications," *IEEE Transactions on Software Engineering*, 22(10): 706-718, 1996.

[87] Maxwell, K., Van Wassenhove, L., Dutta, S., "Performance Evaluation of General and Company Specific Models in Software Development Effort Estimation," *Management Science*, 45(6):787-803, June 1999.

[88] McIver, J.P., Carmines, E.G., Sullivan, J.L., *Unidimensional Scaling*. Sage Publications, Beverly Hills, CA, USA, October 13, 2004.

[89] Mendes, E., Lokan, C., Harrison, R. and Triggs, C., "A Replicated Comparison of Cross-Company and Within-Company Effort Estimation Models Using the ISBSG Database," *Proceedings to the International Metrics Symposium*, Como, Italy, pp. 36-46, 2005.

[90] Meznies, T., Chen, Z., Port, D., Hihn, J., "Simple Software Cost Analysis: Safe or Unsafe?" *Proceedings of International Workshop on Predictor Models in Software Engineering*, May 15, St. Louis, Missouri, USA, 2005.

[91] Mockus, A., Weiss, D.M. Ping-Zhang, "Understanding and predicting effort in software projects," *Proceedings to the 25th International Conference on Software Engineering*, Portland, OR, USA, 3-10 May 2003, IEEE Computer Society, pp. 274-284, 2003.

[92] Moczadlo, R. *Chancen und Risiken des Offshore Development. Empirische Analyse der Erfahrungen deutscher Unternehmen* [Opportunities and Risks of Offshore Development. Empirical Analysis of Experiences Made by German Companies], FH Pforzheim, Pforzheim, Germany, 2002.

[93] Morisio M., Romano D., Moiso C., "Framework Based Soft-ware Development: Investigating the Learning Effect," *Proceedings of the 6th IEEE International Software Metrics Symposium*, Boca Raton FL, 4-6 November 1999, pp. 260-268.

[94] Nazareth, D.L., Rothenberger, R.A., "Assessing the cost-effectiveness of software reuse: A model for planned reuse," *Journal of Systems and Software*, 73:245-255, 2004.

[95] National Bureau of Economic Research, Inc., *Output, Input, and Productivity Measurement. Studies In Income and Wealth Volume Twenty-Five by The Conference On Research In Income and Wealth*, Technical Report, Princeton University Press, Princeton, USA, 1961.

[96] Niessink, F., van Vliet, H., "Two case studies in measuring software maintenance effort," *Proceedings to the International Conference on Software Maintenance*, 16-20 November 1998, Bethesda, MD, USA, IEEE Computer Society, Los Alamitos, CA, USA, 1998, pp. 76-85.

[97] Norden, P.V., "Curve Fitting for a Model of Applied Research and Development Scheduling," *IBM Journal Research and Development*, 3(2):232-248, July 1958.

[98] Noth, T., Kretzschmar, M. *Estimation of software development projects*, Springer Verlag, Berlin, Germany (in German).

[99] Park R., "The Central Equations of the PRICE software cost model," *Proceedings of the 4th COCOMO Users' Group Meeting*, Software Engineering Institute, Pittsburgh, PA, USA, November 1988.

[100] Parkinson, C.N., *Parkinson's Law and Other Studies in Administration*, Houghton Mifflin Company, Boston, USA, 1957.

[101] Parrish, A., Smith, R., Hale, D., Hale, J., "A field study of developer pairs: productivity impacts and implications," *IEEE Software*, 21(5):76-79, September/October 2004.

[102] Paulk, M. C., Chrissis, M.B., *The 2001 High Maturity Workshop*. Special Report, CMU/SEI-2001-SR-014, Carnegie Mellon Software Engineering Institute, Pittsburg, PA, USA, January 2002.

[103] Potok, T.E., Vouk, M.A., "The Effects of the Business Model on Object-Oriented Software Development Productivity," *IBM System Journal*, 36(1):140-161, January 1997.

[104] Putnam, L.H., *Linking the QSM Productivity Index with the SEI Maturity Level. Version 6*, Quantitative Software Management, Inc., McLean, VA, USA, 2000. www.qsma.com/pdfs/LINKING6.pdf

[105] Putnam, L. H., Myers, W., *Executive Briefing: Managing Software Development*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[106] Putnam, L.H., Myers, W., *Measures for Excellence. Reliable Software on Time, Within Budget*. Yourdon Press, Upper Saddle River, NJ, USA, 1992.

[107] *The QSM Project Database*, Quantitative Software Management, Inc., McLean, VA, USA www.qsm.com/database.html

[108] Rico, F., *Using Cost Benefit Analyses to Develop Software Process Improvement (SPI) Strategies*, A DACS State-of-the-Art Report. ITT Industries, Advanced Engineering & Sciences Division, New York, USA, 2000.

[109] Rine, D.C., Sonnemann, R.M., "Investments in reusable software. A Study of software reuse investment success factors," *Journal of Systems and Software*, 41(1):17-32, 1998.

[110] Ruhe, M., Jeffery, R., Wieczorek, I., "Cost Estimation for Web Applications," *Proceedings of the 25th International Conference on Software Engineering*, 3-10 May 2003, Portland, Oregon, USA, pp. 285-194.

[111] Sawyer, S., Guinan, P., "Software Development: Processes and Performance," *IBM Systems Journal*, 34(7):552-569, 1998.

[112] Selby, R.W., "Enabling reuse-based software development of large-scale systems," *IEEE Transactions on Software Engineering*, 31(6):495-510, June 2005.

[113] Sheskin, D., *Handbook of parametric and nonparametric statistical procedures*. 3rd edition, Chapman & Hall/CRC, Boca Raton, FL, USA, August, 2003.

[114] Siy, H., Mockus, A., "Measuring domain engineering effects on software change cost," *Proceedings of the 6th International Symposium on Software Metrics*, Boca Raton, FL, USA, IEEE Computer Society, Los Alamitos, pp. 304-11, 1999.

[115] Smith, R.K., Hale, J.E., Parrish, A.S., "An empirical study using task assignment patterns to improve the accuracy of software effort estimation," *IEEE Transactions on Software Engineering*, 27(3):264-271, March 2001.

[116] Soliman, K.S., "Critical success factors in implementing software reuse: a managerial prospective," *Proceedings of the International on Information Resources Management Association Conference*, Anchorage, AK, USA, 21-24 May 2000, pp. 1174-1175.

[117] Subramanian, G.H., Zarnich, G.E., "An Examination of Some Software Development Effort and Productivity Determinants in ICASE Tool Projects," *Journal of Management Information Systems*, 12(4):143-160, 1996.

[118] Software Technology Transfer Finland (STTF), http://www.sttf.fi/index.html

[119] Teasley, S.D., Covi, L.A., Krishnan, M.S., Olson, J.S., "Rapid software development through team collocation," *IEEE Transactions on Software Engineering*, 28(7):671-83, July 2002.

[120] *CHAOS Chronicles*, The Standish Group International, Inc., West Yarmouth, MA, USA, 2007.

[121] Tomaszewski, P., Lundberg, L., "Software development productivity on a new platform: an industrial case study," *Information and Software Technology*, 47(4):257-269, 2005.

[122] Trendowicz, A., Heidrich, J., Münch J., Ishigai, Y., Yokoyama, K., and Kikuchi, N., "Development of a Hybrid Cost Estimation Model in an Iterative Manner," *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, pp. 331-340, 2006.

[123] van der Pohl, K.G., Schach, S.R., "A Software Metric for Cost Estimation and Efficiency Measurement in Data Processing System Development," *Journal of Systems and Software*, 3:187-191, 1983.

[124] Vaneman, W.K., Trianfis, K., "Planning for technology implementation: An SD(DEA) approach," *Proceedings of the Portland International Conference on Management of Engineering and*

Technology, Technology Management in the Knowledge Era, PICMET-Portland State University, Portland, OR, USA, 2001.

[125] Vijayakumar, S., "Use of historical data in software cost estimation," *Computing & Control Engineering Journal*, 8(3):113-119, June 1997.

[126] Wang, E.T.G., Barron, T., Seidmann, A., "Contracting Structures for Custom Software Development: The Impacts of Informational Rents and Uncertainty on Internal Development and Outsourcing," *Management Science*, 43(12):1726-1744, December 1997.

[127] Yang, Y., Chen, Z., Valerdi, R., Boehm, B.W., "Effect of Schedule Compression on Project Effort," *Proceedings of the 5th Joint International Conference & Educational Workshop*, *the 15th Annual Conference for the Society of Cost Estimating and Analysis and the 27th Annual Conference of the International Society of Parametric Analysts*, June 14-17 2005, Denver, Colorado, USA.