International Journal of Software Engineering
and Knowledge Engineering
Vol. 18, No. 1 (2008) 37–64

World Scientific
www.worldscientific.com

MEASUREMENT IN SOFTWARE ENGINEERING: FROM THE ROADMAP TO THE CROSSROADS

CARLO GABRIEL PORTO BELLINI* and RITA DE CÁSSIA DE FARIA PEREIRA†
Universidade Federal da Paraíba at João Pessoa, Brazil
**bellini@ccsa.ufpb.br*
†*ritacfpereira@ccsa.ufpb.br*

JOÃO LUIZ BECKER
Universidade Federal do Rio Grande do Sul at Porto Alegre, Brazil
jlbecker@ea.ufrgs.br

Received 25 December 2005
Revised 2 September 2006
Accepted 30 November 2006

Research on software measurement can be organized around five key conceptual and methodological issues: how to apply measurement theory to software, how to frame software metrics, how to develop metrics, how to collect core measures, and how to analyze measures. The subject is of special concern for the industry, which is interested in improving practices — mainly in developing countries, where the software industry represents an opportunity for growth and usually receives institutional support for matching international quality standards. Academics are also in need of understanding and developing more effective methods for managing the software process and assessing the success of products and services, as a result of an enhanced awareness about the emergency of aligning business processes and information systems. This paper unveils the fundamentals of measurement in software engineering and discusses current issues and foreseeable trends for the subject. A literature review was performed within major academic publications in the last decade, and findings suggest a sensible shift of measurement interests towards managing the software process as a whole — without losing from sight the customary focus on hard issues like algorithm efficiency and worker productivity.

Keywords: Software measurement; software management; software engineering; measurement theory; complexity; interpretive data; triangulation.

1. Introduction

Historically, companies devote little attention to performance criteria for projects and product development processes [67]. Nevertheless, only through the continuous identification and correction of detours can an organization stand on the com-

petitive edge [18]. In fact, what is correctly measured is correctly managed [43]; moreover, without constant measurement there is no process management, and, with no process management, there are no improvements [5, 50]. In other words, management needs measurement for being accurate, but measurement needs management for having purpose [22]. Additionally, managing and improving relate to two broad processes: *planning* (the *effective* path towards a goal) and *controlling* (the *efficient* path to it) [63].

However, projects with unstable requirements, like those of software, are hard to manage [69]. In particular, software teams are usually assembled afresh for each new project, hindering the development of a shared work history by its members [42]. Moreover, programmers — unlike physicians and engineers, for instance — do not have professional standards to follow [92], and this is likely to be a major source of negative influence over the teams. Software teams indeed constitute a challenge for management [19, 105], and a typical effect is late intervention in problematic projects [69]. As a matter of fact, the software industry deploys only a few metrics from the many available for controlling the development process and predicting product features [84, 110].

With this in mind, our work tries to build a comprehensive view of software measurement, which would lie behind software management. Software measurement has its roots in debates on the efficiency of computer programs and the productivity of programmers, but in recent years the field is heading steadily towards more managerial issues. Such a trend — which can be framed as departing from a concern on hard matters such as the complexity of software algorithms and moving towards softer issues like managing projects involving multi-disciplinary professional teams — mirrors influences the field has experienced from knowledge areas previously kept at a distance, like human resources management, marketing, organizational theories, and research methods.

This paper discusses sequentially the following: the procedures effected for the systematic review on software measurement; the foundations of measurement theory, inasmuch as this theory sets the grounds for any endeavor of planning and control for the software process; the scope of software metrics, recent developments on the subject, and the prevalent method for identifying metrics; common procedures, current debates and presumable trends for collecting and analyzing measures; a comprehensive view of the software measurement field built on the topics previously presented, which exert the greatest potential impact on academic and industry practices; and finally, the exhortation that the research's theoretical developments should be subsequently applied to the software industry in order to see whether there is a match between the state of the art and the state of the practice (a particular application within the Brazilian software industry is currently under way), as well as whether one (art or practice) should precede the other.

2. The Systematic Literature Review

Mapping out the current state of a knowledge field demands a sound methodological effort that should include explicit research questions and quality criteria to be used for selecting the sources of information (the *primary studies*) and the specific concepts to be collected [73]. In this sense in our enterprise we first set the following research questions:

- How have the main concept areas concerned with software measurement developed so far?
- What is being researched at the frontiers on software measurement?
- What are the implications for research and practice from the trends in software measurement?

We started by searching for the areas of interest related to measurement in software engineering — typically the discipline that deals with software measurement [26] — for this relying mostly on the *Guide to the Software Engineering Body of Knowledge* [1]. We thus claimed that discussing software measurement should address the foundations in measurement theory, alternative methods to collect and analyze core measures, the concept of software metrics, and how to identify metrics. From this initial set of concept areas, leading publications related to such areas were selected (journals, books, and technical reports or guides; conference proceedings were not included in the search, due to papers wherein being probably still under construction) based on the opinion of experts and on known rankings within the academic community (e.g., [68, 85]). Table 1 synthesizes the major theoretical sources of information, in which searches followed a systematic routine.

The next step was to search the primary studies for works dealing with the concept areas previously identified — whenever applicable (availability of journal issues), the search started in 1990. Additionally, main references within each publication were occasionally researched, as well as other publications of the authors whose works were reviewed. Aided by electronic search engines, we started by looking for the expressions in Table 2 within the articles' titles. Although these expressions were preferred beforehand, a complete search in all articles was done in order to possibly find similar expressions in the titles, as well as other concepts not included in the original search — that served merely to assemble a minimum set of relevant primary sources. Two of us were then assigned to reading the resulting set of articles after the relevance was confirmed from their abstracts and the conclusions.

After the selection of the primary sources, we followed a "bottom-up" approach to content analysis for building the categories of interest: the set of categories was developed as the study unfolded from the primary sources, so we devised the whole picture of software measurement only after concluding the readings. Although bottom-up approaches may lead to fortuitous schemes [53], we had no better framework upon which to develop the categorization; moreover, it was our very intent to let the primary sources drive the research as independently as possible from our

Table 1. Sources for the systematic literature review.

Category	Title
Journal	*ACM Transactions on Software Engineering and Methodology* *Advances in Engineering Software* *Communications of the AIS* *Empirical Software Engineering* *IEEE Transactions on Software Engineering* *Industrial Management and Data Systems* *Information and Management* *Information and Organization* *Information and Software Technology* *Information Systems Journal* *Information Systems Research* *International Journal of Human-Computer Studies* *International Journal of Technology Management* *Journal of Management Information Systems* *Journal of Systems and Software* *Journal of Systems Management* *Journal of the Operational Research Society* *Management Science* *Measurement* *MIS Quarterly*
Book	*Applying Software Metrics* [87] *Essentials of Project and Systems Engineering Management* [39] *Rethinking Management Information Systems* [30] *Software Engineering* [97] *Software Engineering* [110]
Report/Guide	*Guide to the Software Engineering Body of Knowledge* [1]

assumptions. The discussion of the sources and the categories that emerged are presented in the following sections, organized around the five key areas for framing software measurement.

3. Measurement in Software Engineering

Measurement is essential for science [34], and in organizations it serves to help manage by fact, not by feeling [33]. In software engineering, it still lacks consolidated terminology, principles and methods [2, 103], but it is said to address processes, products and resources [26] and to be useful for (1) nourishing visibility and understanding, (2) establishing the grounds for improvements, and (3) planning, monitoring, and controlling processes, products and resources [95]. It is also well accepted that software measurement activities include direct and indirect assessments, as well as predictions [26, 44, 64, 110]. Table 3 illustrates software measurement interests.

Measuring software involves knowing how to deploy *measurement theory*. In fact, this theory, a branch of applied mathematics [104] rooted in developments made

Table 2. Searches within the articles' titles.

Expression	Possible results (Examples)
"analy"	"[data/measure] analysis", "analyze", "analytical"
"assess"	"[software] assessment", "assess"
"collect"	"[data/measure] collection", "collect"
"control"	"[process] control"
"develop"	"[software] development", "develop"
"eff"	"effectiveness", "efficacy", "efficiency"
"error"	"[measurement] error"
"instrument"	"instrument", "instrumentation"
"manag"	"management", "manage"
"measure"	"measure", "measurement"
"method"	"method", "methodology", "methodological"
"metr"	"metrology", "metric"
"plan"	"[process] planning", "[software] plan"
"predict"	"predict", "predictive", "prediction"
"process"	"[software] process"
"quali"	"quality", "qualitative"
"quanti"	"quantity", "quantitative", "quantify"
"valid"	"validation", "validity", "validate"

during the 19th century but only truly matured in the last five decades or so [35], is consistently developing in software engineering [24, 72]. It is closely related to Stevens' *theory of scales* [64, 81] and basically involves setting unequivocal relations between an empirical measurement object and a symbolic system representing some attribute of it that is of interest for measurement [23, 24, 26, 81, 102, 104], in order for one to access the "real world" object by means of processing symbols equated to its attributes [81] and reducing biases introduced by measurement error [106]; nevertheless, errors of a statistical nature — like the random measurement error — are not of concern to measurement theory, and it is also taken for granted that measurements are always discrete — that is, they exhibit limited precision [104]. The following non-exhaustive concepts are essential for framing the theory [23, 24, 35, 64, 104]:

- *empirical relational system* (ERS) — qualitative description of objects, relations and operations representing the portion of reality where measurement takes place, as well as the extant empirical knowledge about attributes of the objects one wants to measure;
- *formal/symbolic relational system* (SRS) — description of the domains for the measures on the objects' attributes, as well as the relations of interest between measures; systems ERS and SRS are linked by means of measures and scales (discussed below);
- *measure* — formal mapping between the two systems, matching ERS elements

Table 3. Examples of software measurement interests. Source: adapted from [45].

Entity	Attributes	
Product	**Internal**	**External**
Specification	size, reuse, modularity, redundancy, functionality, syntactic correctness	understandability, maintainability
Design	size, reuse, modularity, coupling, adherence, inheritance, functionality	quality, complexity, maintainability
Coding	size, reuse, modularity, coupling, functionality, algorithm complexity, flow of control	reliability, usability, maintainability, reusability
Test data	size, range	quality, reusability
Process	**Internal**	**External**
Development specification	time, effort, number of changes in requirements	quality, cost, stability
Detailed design	time, effort, number of defects in specifications	cost, cost effectiveness
Test	time, effort, number of defects in coding	cost, cost effectiveness, stability
Resource	**Internal**	**External**
Personnel	age, cost	productivity, experience, intelligence
Teams	size, level of communication, structure	productivity, quality
Organizations	size, ISO certification, CMM level	maturity, profitability
Software	price, size	usability, reliability
Hardware	price, speed, memory size	reliability
Offices	size, temperature, light	comfort, quality

with SRS numbers/symbols and observing the equivalence of relations between the systems;

- *admissible transformation* — transformation that preserves the equivalence between empirical and symbolic relations;
- *nominal scale* — strictly one-to-one admissible transformations allowing exclusively the empirical relation "equality";
- *ordinal scale* — admissible transformations strictly on an increasing monotonous function allowing exclusively the empirical relations "equality" and "order";
- *interval scale* — positive linear admissible transformations ($f(x) = ax + b, a > 0$) allowing exclusively the empirical relations "equality", "order" and "difference";
- *ratio scale* — positive similar transformations ($f(x) = ax, a > 0$) allowing the empirical relations "equality", "order", "difference" and "relative difference"; and
- *absolute scale* — no transformation is meaningful except the identity ($f(x) = x$).

With the rigorous approach provided by theory [26], measurement in software engineering is made easier to frame and manage. In particular, there is the need to deepen and systematize our comprehension about the attributes of the objects of interest, which will then give rise to a theoretical and formal system with which

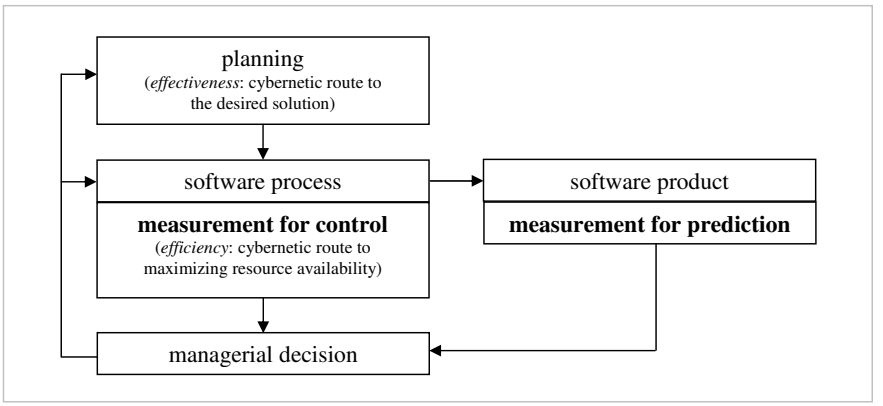
an object and its attributes would be subsequently dealt with. This leads one to address the more fundamental issue of understanding the concept behind the object and the attributes, adopting a particular perspective for working with them and standardizing it (for a case in software quality, see [64]).

Although measurement issues seem trivial at first, each concept may convey a great deal of reflection and heated debate (e.g., [80, 81, 107]. The most fundamental discussion is whether the object under empirical examination has attributes that are conceptualized by current theories and that can be measured by the available methods (that is, whether the values that are produced by measuring the object's attributes are within known value ranges and really reflect the nature of the object). This exerts direct impacts on the purpose and on the use of measures. Our perspective joins that of [81] and assumes that measures do not relate to "actual" (intrinsic) attribute values (or *true scores* [34]), but to outcomes of procedures currently deemed appropriate for getting purposeful information about real-world objects; that is, given the likely endless philosophical debate on an object's ontology, interpretation and subjectivity are in fact the very intrinsic ingredients of every measurement attempt, and this helps explain why there should be a sound conceptual framework underlying the software measurement endeavor.

Recent advances in measurement theory indicate the need for a probabilistic version of it [102]. It is also of current concern in the discussion around a more pragmatic and flexible deployment of scales and corresponding statistical procedures [23, 24, 64], like in what can be called a *weak measurement theory* [83]; in some cases, like in studies on the relation between information technology and organizational dimensions, less rigid scale transformations have been already frequently performed — admissible transformations are indeed more of a mandatory (rather than exclusive) nature [104]. And in what comes to specific deployments of measurement theory to software engineering, bold developments are expected in the field of quality [64].

Critiques were unveiled to applying measurement theory to software engineering, since the theory would only provide the means for handling a set of classic measurement issues [2]. The broader field of *metrology*, which covers theoretical and practice-oriented issues alike, should in their view be considered when setting the basis for developing and applying measurement instruments and processes. Metrology would look after defining measurement principles, which in turn would help negotiate methods and procedures for measuring. Symptomatically, current initiatives in software engineering would be lacking a consistent approach to effectively address the instrumentation for measurement.

Irrespective of how one frames the deployment of measurement theory to software engineering, its application should provide the means for developing measures independently of whom is in charge of the process, as well as measures that address solely the empirical object of interest [81]. Furthermore, the outcomes of measurement must adhere to assessing and predicting the quality of products, processes and resources.



Source: adapted from [15], [18], [63], [92], [101], [110].

Fig. 1. Measurement for software control and prediction.

4. Software Metrics

Software engineering and metrics are bound together [44, 54, 72]; in fact, metrics constitute the dominant approach to measurement in software engineering [2]. In [92] and [110], metrics are said to relate to process control — like the average effort and time demanded when fixing defects — and to the prediction of product features — like the number of operations associated to an object. More broadly, metrics are key for vigorous research [114], serving as feedback and measurement tools for assessing whether one is proceeding correctly [29], as well as drivers for engineering and management processes [54]. They are organic to the software process [39, 54] in the sense that they support information system (IS) managers in estimates, technical tasks, project control, and process improvement [97]. In particular, metrics are the only factor currently available for contrasting companies in terms of process maturity [98]. Figure 1 synthesizes how metrics apply to software development.

The key for the effectiveness of metrics is the development of a metrics plan describing *who/how* (tools, techniques, and personnel), *what* (is to be measured), *where/when* (in the measurement process) and *why* metrics [95] (after all, they must be useful [75]). At the same time, the abstraction level of measures should be addressed for building any metric, since not always — or almost never — it is possible to measure software quality attributes directly; building unidimensional measures is truly the outcome of robust theoretical and statistical modeling [28, 107]. Likewise, the aggregation level of the work system of a software organization (e.g., business unit, project, or component) should also be taken into account when making the metrics plan [75]. The result is that the surrounding context of measurement must be carefully assessed, given that software projects usually involve variables of a highly dynamic, complex nature and presenting fuzzy relationships with other variables [9].

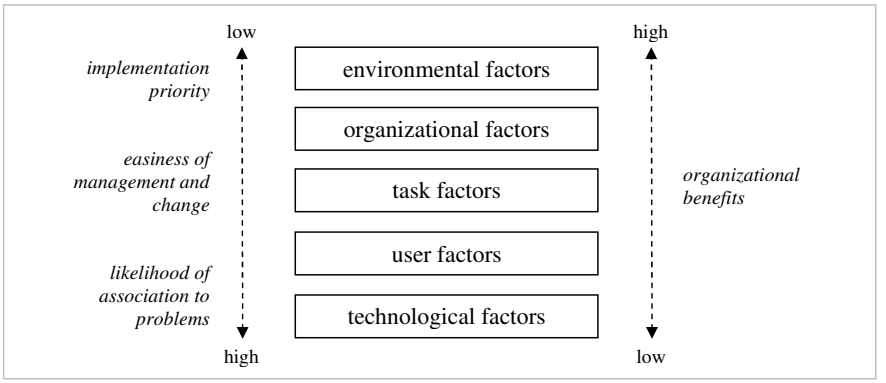
Committing oneself to a metrics agenda means to be prone to change towards a culture in which decisions are made based on relevant, accurate, practice-oriented data [62]. This is in line with the assumption that the only rational way for improving any process is by measuring specific attributes of it, developing a set of metrics appropriate for the attributes and applying the metrics to provide signals of improvement [97]. Moreover, choosing metrics, collecting data, discussing the results of measurement and taking due action take up time as well as other nontrivial resources, and this only makes sense if such activities address specific improvement goals [95]. Nevertheless, little is said about the successful implementation of metrics in the realm of software process improvement [44, 62].

There is, however, some inadvertent use of terms like "measure" and "metric" in the literature [118], the reason why we adopt and make explicit the conceptualization in [97]:

- *measures* result from computing data from a software project, process or product, and indicate in quantitative terms the magnitude of an attribute;
- *metrics* result from computing measures, and indicate in quantitative terms the degree to which a system, component or process exhibits some attribute; and
- *indicators* result from computing metrics, and help us to develop insights on software projects, processes and products.

A number of software metrics have been proposed over the years for a myriad of interests, but sometimes complementary or conflicting rationales and empirical evidences were assumed between works. The kingdom of metrics is indeed large and complex, so in this short paper we present a general picture of some illustrative cases. For instance, source-code metrics are among the most popular in some scientific communications and industry practices [44], like metrics for algorithm complexity and size; nevertheless, no current complexity metric addresses completely what is needed for controlling, managing, and maintaining software [27], and metrics of this kind exhibit obscure relationships with software quality [110] and programmer productivity [52]. On the other hand, attention is increasingly being paid to multidimensional metrics addressing the whole software endeavor, like those on process and project management (e.g., [88]). In fact, it was already demonstrated that, during IS implementation, heterogeneous factors play a role [108]. In Fig. 2, factors influencing the software process are said to be in reverse order to their implementation priority in practice. This means that the most important factors (higher in the hierarchy) are seldom implemented, which is an explanation for why many projects fail [60, 96, 109, 111], as well as why technological issues (appearing at the bottom) are the first — and sometimes the only — concern in projects.

Figure 2 is rich in insights for understanding why so much attention is given to technical attributes in projects (and, therefore, to technology-oriented metrics). First, since the hierarchy is based on Maslow's hierarchy of needs, in which higher-level needs are only addressed when lower-level needs are satisfied, the fact that technology is the dominant preoccupation in IS projects may mean that the



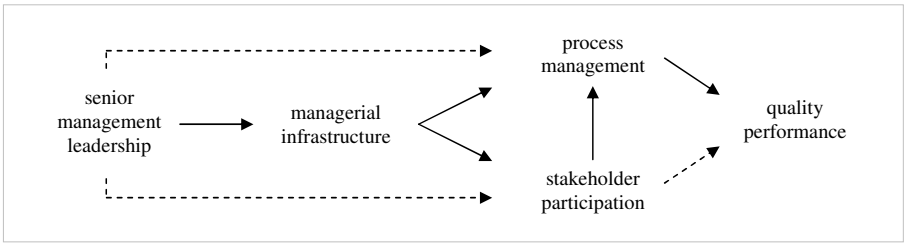
Source: adapted from [108], reproduced from [15].

Fig. 2. Hierarchy of factors affecting IS implementation.

implementation of technology is currently ineffective (be it due to the technology itself or to its application), thus preventing attention to be paid to higher needs. Second, technology being the most desirable dimension for effortless management may have an influence on developers not being incited to take care of other dimensions of the solution. Third, the natural tendency of connecting deficiencies in implementation to the likelihood of lower-level needs not being fully satisfied perhaps favors an approach whereby excessive attention is devoted to perfecting — maybe endlessly — the fulfillment of the more fundamental needs. Other explanations may relate to particular preferences of developers and the institutionalization of practices in the profession and the industry.

Sound developments aimed at changing this state of affairs were recently made in a research that compiled and extended critical social issues for the management of customer teams during the co-development (with technology consultants and suppliers) of software for one-of-a-kind enterprise information systems [14]. That endeavor addressed the teams' structure/organization and the personal traits of the teams' professionals (that is, the *social subsystem* of the socio-technical approach to work design), and it ended up proposing a set of seven indicators, 27 metrics and 88 measures that, together with current process/task and technology criteria (the *technical subsystem*), help managers design, control and assess the teams. Although targeted at a specific subset of social measures in the software field, such an accomplishment works out factors largely neglected in research and practice.

According to [44], among the key trends and needs of the metrics field, one can find a deeper approach to uncertainty and to combining heterogeneous subjective evidences, as well as some disregard to the traditional regression analyses — which may obstruct a fuller understanding about causality. They also talk about advancements in meta-analyses, mainly on (1) the mechanics of metric implementation programs, (2) the deployment of metrics in empirical software engineering, and



Legend: continuous arrows indicate direct, positive effect; dashed arrows indicate indirect, positive effect.
Source: [100].

Fig. 3. Metrics domains for the software endeavor.

(3) theoretical foundations of software metrics. There is also room for discussing further challenges:

- maybe most academic research is not relevant in substance nor in scope for the industry;
- little is known about the effectiveness of metrics (sound improvements are made in [54]); and
- little is known about the true reasons why, notwithstanding current critiques, metrics like lines of code, defect counts, cyclomatic numbers and function points still have their place among the most popular standards.

Another arresting theme in software metrics is introducing to measurement the very context in which measurement occurs [25]. In this sense, it is in increasing obsolescence assessing software with no explicit regard to the environment in which the software is handled; after all, choosing a particular project design gives rise to inevitably circumscribing the quality attributes for the software [20]. There is indeed some exhortation that the technical validation of a system should be performed only after the validation of its very context [82], but this seems not trivial to understand nor to effect.

What is clear, though, is that a comprehensive, quality-oriented management — by means of metrics — of the software endeavor is in need. According to [100], the key components of an organizational system oriented towards such a goal (product quality and process efficiency) are (see also Fig. 3):

- *senior management leadership* — degree to which senior IS management sponsors improvements on quality and theorizes on quality initiatives for the systems development organization;
- *managerial infrastructure* — structural property of the IS organization related to creating an organizational setting oriented towards quality for the central processes and work practices;
- *process management* — degree to which the key project and development processes get defined, controlled and systemically improved; and

- *stakeholder participation* — degree to which the work practices are set in a way that each group contributes with complementary knowledge to the other groups involved.

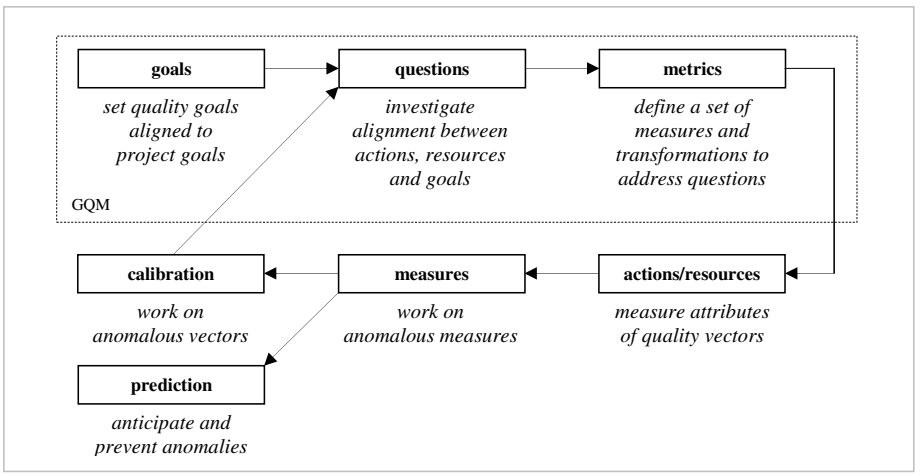
5. Identification of Metrics

Measuring software should head steadily towards appropriate metrics for each application context and directions on how to apply them [49], in order not to give rise to unintended side effects [33, 112]. This challenges interpretations that some measurement is better than none (as in [52]). The first and perhaps most important procedure is to choose the measures that should be effected; after all, the outcome of this decision propagates throughout the sequence of activities and is intrinsically related to the very goals of the measurement system. From [58] and [118], one can organize the following components for any measure:

- *handle* — an appropriate name for the measure;
- *description* — what the measure is and what are the goals of collecting it;
- *relationship* — how will the measure be related to the software process;
- *history* — what we know about the measure from previous experiences;
- *expectation* — what is expected from the measure in the future;
- *source* — where the measure is to be collected;
- *tools* — technology available for measurement support;
- *observation* — how will the measure be collected;
- *frequency* — when will the measure be collected;
- *stakeholders* — who will be involved with measurement;
- *scale* — measurement units;
- *range* — maximum and minimum values to be observed;
- *threshold* — control and triggering values for measurement;
- *validation* — data for validating relationship strength and accuracy;
- *interpretation* — how will the measure be screened;
- *report* — measurement documentation; and
- *actions* — events triggered by the measure.

Therefore, systematizing the selection of measures is crucial for the success of metrics, and, according to mainstream, leading publications in software engineering, one can safely conclude that the Goal-Question-Metric (GQM) approach [11] is the dominant alternative to doing it. In fact, GQM is frequently deployed in its original or adapted formulation, or even combined with other models (e.g., [23, 25, 26, 75, 76, 95]). GQM defines, institutionalizes and systematically addresses measurement programs that support the quantitative assessment of software products and processes [49]. GQM involves the following:

- the design of corporate, division or project goals usually targeted at productivity and quality issues and always tied to the organization's ultimate goals;
- the development of questions that sharpen the goals, in order to make visible



Source: adapted from [110], [11].

Fig. 4. Integrating GQM to measurement.

whether they were fulfilled or not, by means of identifying uncertainty points related to the goals — the underlying assumption is that reaching any goal involves answering specific questions; and

- the identification of metrics related to measures to be collected which answer the questions (deriving measures from questions which in turn are rooted in superior goals gives GQM its top-down character).

Figure 4 shows how GQM can be the cornerstone for a measurement program.

Among the GQM variants, it is worth mentioning the work of [25], which was supported by one of GQM's fathers and extends the original framework. They developed the GQM/MEDEA (GQM/MEtric DEfinition Approach) model, which combines GQM to empirical hypotheses (empiricism is in fact present in many recent developments and aspirations in software engineering [44, 45, 49, 54]); empirical hypotheses then undergo experimental verification based on expected mathematical properties of the theoretical measures from the attributes of interest. The most important contribution of GQM/MEDEA to the original framework would be establishing a systematic process to defining software product measures from GQM goals. Moreover, the model makes explicit all decisions involved in planning the measurement activity for building a predictive model; in fact, this recent model emphasizes making directions — maybe in reaction to previous critiques from the literature (e.g., [58]). The experimental-empirical character is also made apparent when the authors name the mandatory traits of an effective software measurement process (which would be integral to GQM/MEDEA): the process must be disciplined, rigorous and based on goals, with properties and experimental assumptions accurately defined, and the process must also include comprehensive, experimental

validation. Synthetically, the new model addresses:

- a detailed description of the activities involved in defining the measures and informational flows between activities;
- the integration of the definition of measures, the corporate goals and the development context;
- the fact that measures should not be defined *per se*, but according to the theoretical context;
- the support for the rationale, interpretation and reuse of measures;
- the support for identifying problems that may occur during the definition of measures, taking into account that the process is "human" intensive; and
- a conceptual model to be deployed in the implementation of a repository of relevant knowledge for measurement.

GQM is not free from critiques. First, top-down methods are not easily espoused by those who are assigned to implement it; while it is relatively straightforward to set goals and develop the questions, it is extremely hard to measure them effectively, and conflicts between developers and managers may then be easily brought into existence [58]. The model also lacks an approach to connecting metrics that answer the same question, as well as to guiding one through implementing the measures [95]. Finally, GQM's top-down character bypasses the actual measurement needs that are known only at the bottom [58].

Bottom-up models for identifying metrics are also available, starting from measurable items and then building management goals based on the measures. Such models build a database of measures to be collected on each product according to the following [58]:

- *input measures* — information about resources (personnel, computers, tools, etc.), processes and activities performed;
- *output measures* — information about the production outcomes; and
- *result measures* — information about the use and the effectiveness (perceived and actual) of the production outcomes in fulfilling requirements and satisfying people.

What is behind the bottom-up approach is that the primary function of measurement is to support the engineering tasks by raising questions and helping rich insights to develop.

6. Measuring

Collecting and analyzing the measures are two fundamental processes of software measurement intrinsically related to the previous discussion on measurement theory — which provides the means to formalize such processes, a requisite for an effective metrics plan. Given the intensive human involvement with software processes and products, the instrumentation for collecting and analyzing measures in software

engineering owes much to methodological advances in the social sciences [24, 25, 49]; as a matter of fact, software development is socio-technical in nature [37].

6.1. *Assembling the evidences*

Collecting the measures — or what we here call the process of assembling the evidences — is the way to efficiently gather data that provide the highest predictive power for a set of stated goals [79]. According to [58], the collection of measures should follow the principles of (1) not being obtrusive, (2) being automated whenever possible, (3) being based on public, explicit, unambiguous definitions, (4) validating measures as soon as they are collected (and as close to the source as possible), and (5) integrating the collected measures to a repository for future validation. A robust collection of measures is vital for the empirical software engineering field, since this field aims to build a reliable base of measures for professionals and researchers (Votta *et al.* mentioned in [49]). Important it is, however, that the set of measures be valid (represents what it is intended to represent [40]), inasmuch as no amount of data neutralizes a poor isomorphism between theoretical *constructs* or *latent variables* and the methods used to measure them [34]. For this, the largely neglected discipline on instrumentation for measurement in software engineering demands greater care [2].

The process of collecting measures should be adapted to each organization [119] and the subsequent implementation may take different forms — it depends on the nature of the measures (in regard to an object's attributes), on the researcher's knowledge about what is to be measured, on the purpose of collecting, and on the appropriateness of the instruments. Thus, measurement may be direct or indirect [64], objective or subjective [58, 95], based on theory or not [34], applied to people or objects, and consist of different sorts and quantities of data [23]; but it should not be left out of sight that measurement ought to be as focused as possible, that is, collect exclusively the intended data that answer purpose-driven questions [118].

All this leads to the variety of methodological procedures available. A comprehensive review of collection techniques, however, is not workable in this piece of text, so we next make reference to current literature discussions on measure collection with some emphasis on directions from the *Guide to the Software Engineering Body of Knowledge* [1]. Another decision made in this paper for the discussion to be more straightforward and focused on current developments was to exclude collection methods like automatic algorithm analysis (for measuring size, coupling, etc.) or computational mechanisms for knowledge discovery in databases. Such a decision took into account the fact that software engineering is steadily incorporating methodological developments from the social sciences [24, 25, 49] and that, for software process improvement, organizational factors — measured with the support of corresponding methods — are at least as important [38]. Finally, we opted not to comment specific software packages developed for particular collection procedures, like the increasingly available tools for automatically assembling measurement instruments or performing Web-based data collection.

A current debate deals with the measurement instruments themselves. Notwithstanding the wealth of measures available and their potential applications, little discussion is effected on measurement instrumentation for software engineering [2]; moreover, directions found in the literature or in practice are often misleading. In fact, [23] and [24] criticize usual directions for being inflexible when constraining *a priori* measures to particular scales (like nominal, ordinal, or ratio), since there would be only a few cases where a given measure would be unequivocally tied to one specific scale. As previously stated, scales and admissible transformations (data analysis procedures) are closely related; but, due to the rigid direction of scales being univocally assigned to each measure *a priori*, data collection (and subsequent data analysis) is forced to operate in strict, narrow limits that may subsequently prove to be conservative in excess. A handy argument for advocating the contrary is that it is always possible to move from a stronger to a weaker level of measurement [104]; that is, if a stronger scale (say, ratio) is found incompatible with an object's attributes, it is possible to convert measurements to a weaker scale (say, ordinal). Therefore, it is suggested that instruments for data collection do not follow inexorably early scale assignments to measures.

Another subject of growing interest is the deployment of methods to collect qualitative data; in [58], data collection includes gathering individual perceptions, for instance by means of *surveys* (with structured questionnaires applied to statistically significant samples) or *in-depth interviews* (with semi-structured interview protocols applied to select individuals). Techniques for group discussion such as *focus groups* (small groups of peers assembled for the lively debate of ideas) are also of interest for qualitative measurements [13]; the *expert estimation* [12, 42, 65, 66, 119, 120] is of special concern — in what comes to estimating the effort in software development, qualitative expert estimation is indeed the dominant approach [65].

The instrumentation for data collection is decisive in these cases — similarly to what was previously said about selecting appropriate scales for measurement. For instance, when using instruments for in-depth interviews, face and content validation are of need [21, 28, 61] before and after pre-tests [113]. Notwithstanding the collection of measures in such a case having no statistical meaning, the collection only ends when the answers of a number of individuals converge to the same constructs. In surveys — like for collecting statistically significant information about customer satisfaction with a given software package — instrument development also requires expediency in validation procedures [6]; that is, surveys require rigorous statistical analysis for defining appropriate samples and — additionally to face and content validation, and pre-tests — pilot tests and construct validation (convergent, discriminant, and nomological validation [7, 21, 28]) need to be on the agenda (construct validation, however, may depend on the research's purpose and stage).

Data collection for developing metrics may also be done with direct searches in databases [41, 58, 119, 122], what is best known as the *collection of secondary data* [89]. Moreover, building and maintaining a *metrics repository* [57] or *experience base* [84] is an important decision for future projects [70, 110], mainly when the

Table 4. Methods for collecting measures in software engineering. Source: adapted from [89].

IS Methodology/Method	Proposition in Software Engineering
Library research	Review on the history, the state of the art, and the state of the practice of measures and measurement procedures, as well as on the changing knowledge about the measurement objects.
Literature analysis	Meta-analyses for consolidating terms, methods in particular applications, and objects of interest for measurement.
Case study	Qualitative, longitudinal analysis of the behavior of an object's attributes.
In-depth interviews	Semi-structured interviews for capturing the perception of select experts about the nature of an object's attributes.
Survey	Structured interviews for capturing the perception of a sample of practitioners about the behavior of an object's attributes.
Laboratory experiment	Experimental control over an object's attributes being measured during simulation.
Field experiment	Quasi-experimental control over an object's attributes being measured in real life.
Secondary data	Search in databases of measures collected by previous purpose-driven, comparable processes, in order to understand the behavior of an object's attributes and the correlation with other measures.

organization is committed to learning from previous enterprises [121]; but keeping the database relevant through continuous assessments is also needed [40]. Lastly, the deployment of less usual methods in software engineering, like case studies, may be a strategy [58].

As a contribution, in Table 4 we adapt the categories in [89] for research methods in the IS field, indicating which could be used (and the main concerns of each) for the collection of measures in software engineering.

6.2. *Producing the evidences*

The analysis of measures — or what we here call the process of producing the evidences — comes straightforward from their collection. Exhibiting the stigma of hard work [110], it is nevertheless usually handled with insufficient care [57, 58], notwithstanding the sound benefit to be perceived if performed within efficient feedback mechanisms that leverage organizational processes [54].

Before the actual analysis, collected measures sometimes must follow a "purifying" path. Although the collection of measures should be ideally based on a deliberate process supported by appropriate techniques, the measures may not be ready for analysis. In fact, formatting the data into a friendlier, scale-adherent design should precede measure interpretation [58]. For instance, after fulfilling audio recordings during in-depth interviews (as for measuring initial perceptions on the performance of changes effected to the software process), the discourses should be formatted into appropriate media for particular content analysis procedures [8] to

be later deployed. One alternative is to reproduce the measures into a word processing application or codify them into a package for semantic, quantitative analysis [47]. In either case, the transcription must adhere to the interface with the target medium and — more importantly — follow some relation prescribed by measurement theory governing the equivalence between the recorded items in audio format and the text being built. From this example, we should observe that moving measures between media for analytical purposes conveys also a collection process that must comply with all the previous assumptions from measurement theory — since an accurate transcription of items between representation systems is envisioned, with no interference from deploying the instrument for data "collection".

Developments made in the field of statistics are incorporated by software engineering whenever appropriate [24], and as such a laborious process is also expected when preparing the measures for multivariate analysis. Before choosing and applying the most appropriate methods available (see [55] for a comprehensive review), tests like the ones for sample adequacy and adherence to particular probability distributions should be performed. Such procedures are here not regarded as of a genuine analytical nature (although they truly support the analyses), since *per se* they do not add information to knowledge nor to decision making based on the measures effected — the ultimate purpose of software metrics [58]. Those procedures are more oriented towards supporting simulations, exploring scale attributes, or indicating the need for changing the set of measures (e.g., excluding values or fitting the data to certain statistical procedures in sub-samples), thus guiding the analyst throughout the mathematical maneuvers and conceptual reasoning. What truly distinguishes the analysis is not the deployment of one mathematical method or another, but the incorporation of theory and subjective interpretation to what is measured — which is then ready for scientific scrutiny.

Like what happens for the collection process, formatting the measures and analyzing them rely on numerous factors, among which the type of data collected and the purpose/nature of the analyses [25, 41]. Nevertheless, in all cases measurement theory is helpful for accuracy, conceptual consistency, and process objectiveness [24]. Methodological rigor and pragmatic usefulness thus resultant will root the effective application of metrics for the purposes in mind [54].

An important discussion regarding the analysis of measures concerns the very deployment of measurement theory and the various types of scales (e.g., [23, 24, 26, 64]). As mentioned before, the theory predicts particular relationships between scale types and measures, according to the nature of the latter, and each type includes admissible and non-admissible transformations. Naturally, if we assume that one can hardly seize *impromptu* the nature (and the scale type) of a measure, then it is also true that an arbitrary decision made before the analyses (during the collection of measures) will block a whole set of transformations potentially suitable for the measures (Table 5 reproduces some scale types and corresponding directions). Nevertheless, a more pragmatic standpoint is championed in [23], reckoning the chronic doubt on the nature of some measures and allowing — not without explicit

Table 5. Examples of relating scales to statistics. Source: [23].

Scale Type	Appropriate Statistics (Examples)	Type of Appropriate Statistics
Nominal	mode frequency contingency coefficient	nonparametric
Ordinal	median Kendall's tau Spearman's rho	
Interval	mean Pearson's correlation	nonparametric and parametric
Ratio	geometric mean coefficient of variation	

and meticulous speculations about possible side effects — greater freedom for the analyst to handle them. Such a flexible stance, although not being clearly the mainstream perspective in software engineering, is widely espoused or at least accepted as producing minor hurdles in other knowledge areas — in which one frequently finds, for instance, the computation of arithmetic means for Likert scales. In fact, [104] puts that "there is no need to restrict the transformations in a statistical analysis to those that are permissible", but also "an appropriate statistical analysis must yield invariant or equivariant results for all permissible transformations".

Another debate deals with how causes and effects are investigated. One issue is that correlations between measures (which, once detected, enable one to perform a whole set of statistical procedures and conceptual inferences) merely suggest the simultaneous occurrence of those measures — and *nothing* about antecedents and consequents [44]. Therefore, studies in which nomological networks of constructs are not exhaustively investigated with advanced methods like *structural equations modeling* [51] are not able to postulate genuine causal reasoning.

The authors in [44], with their developments on Bayesian belief networks, claim to have made progress on this regard (causes and effects) and other delicate subjects like the role of uncertainty and lack of information (measures). The authors' model was implemented as a graphic tool which makes transparent to the user the complex Bayesian mathematics behind the intrinsic propagations of probability. The tool takes as input values the outcomes of measurement and depicts the effects from the measures in the complex relationships between components (previously modeled). Summarily, the benefits from using such a model include:

(1) the explicit modeling of "ignorance" and uncertainty in estimates, as well as cause-and-effect relationships;
(2) the unveiling of assumptions originally hidden;
(3) reasonable predictions even with the lack of important data;
(4) *what-if* analyses;

(5) the deployment of probability distributions objectively or subjectively derived; and
(6) rigorous mathematical semantics.

Several other studies make comparisons or devote critiques to particular statistical methods, usually those that can be applied to multiple variables simultaneously (e.g., [13, 25, 56, 71, 119]); but since no clear alignment between those studies was unveiled in our literature review, the present paper does not delve into them.

Finally, regardless of the method, the outcomes from the analyses must be fed back to the organization and especially to the point where measures were collected [58]. Such a feedback should be understood as *lessons learned* to integrate the repositories of metrics [57] and project knowledge [121], in order to serve as a source for future processes of measure collection and analysis.

6.3. *The crossroads in the future of software engineering*

Software engineering faces a critical decision in what comes to its role in the collection and the analysis of measures: in spite of the steady incorporation of methods from other knowledge areas like the social sciences, we find but a few publications (mainly targeted at the IS audience) where instrument validation is of real concern. If the field aims to understand, apply, and become the reference in the canons of software measurement, much more than current exegeses are needed; academic practice should be improved or at least communicated in detail.

Additionally, it is unclear whether software engineering and information systems will remain separate or merge into a single domain in the near future, not only in what comes to measurement issues (concepts, objects of interest, procedures, and so on), but also in terms of the required expertise for its professionals in academy and industry. Take, for instance, new development approaches like the *agile method* of *eXtreme Programming*: its assumptions on informality, prototyping, and pair programming [99, 117, 46] have been typical in research on human-computer interaction, IT-business alignment, and inter-rater validity; the assumption that the agile practices should be implemented as an indivisible whole is not in line with the needed operational flexibility in industry [3], which is a design rule at least since the contingency approach to management came about in the late 1960s [36]; and the espoused practice of avoiding documentation is clearly against good project principles [91]. Further challenges in the field (like performing meta-analyses, promoting the need for empirical studies, and setting the principles for experimental replication) can be found elsewhere [10].

7. Conclusions

Measurement is of primary importance for organizations, supporting them in initiatives for improvement and superior business performance [18, 43, 50, 5, 67]. Notwithstanding, measurement is a complex endeavor involving multiple dimensions of the

objects under analysis, as well as sometimes incongruous conceptualizations from measurers [90]. Remarkably in the software field — software here also understood as a component of information systems [111], which pervade the modern organization — measurement was traditionally dealt with from a shortsighted perspective, focusing mainly on hard issues like algorithm complexity and programmer productivity; nevertheless, it is well-known that software development is socio-technical in nature [37]. Lately, however, software engineering started to face profound changes in its framing of the measurement activity; effects of such a new realm span heterogeneous dimensions of measurement, like the objects of interest — for instance, the context in which a system should be effective in future — and the methods for collecting and analyzing measures. Concerning the new methods, it is of note that the social sciences are increasingly serving as a source of methodological treatments to be applied to software processes and products [24, 25, 49], and a noisy debate about the statistical bases of current methods is under way.

For practice, it is critical that software measurement still makes extensive use of orthodox measures — like those of algorithm complexity and programmer productivity — but also that the field is deeply interested in incorporating developments from research fronts like the organizational studies, marketing, human resources management, and, foremost, research methods. In this sense, project management and individual issues — like customer satisfaction, personnel development, and work environment and climate — are increasingly the focus of research thrilled by the foreseeable benefits of managing the whole software endeavor. Practitioners are asked to contribute with experimenting the new metrics in real-world projects (if they do not do it already), in order to gauge their effectiveness and effects on process efficiency, product quality and social responsibility, as well as to test whether important constraints like schedule and costs are not inadvertently overrun.

Particular directions concern the development and application of instruments for collecting measures. First, practice-oriented measures should be developed consistently, that is, based on a robust and agreed-upon belief on the effective relation between the empirical and the symbolic relational systems. They must also address exclusively the empirical object of interest and be as independent as possible from the measurer. And second, professionals should not feel inhibited to deploy scale transformations for the purpose of supporting the analysis of collected measures whenever the nature of the objects' attributes is not clear enough and given that such transformations seem to purify one's comprehension. This is not to say that freewill rules, but that real-time business demands (like from customers, employees, partners, or the technology) must correspond to, if not optimal, at least discretion-led, satisfactory outcomes from operations and management. Summing up, and inasmuch as a change in culture is integral to the implementation of metrics [62], positive instrumentation should support the software organization.

It is clear that our research's findings need to be surveyed in the software industry in order to see whether there is a match between the art and the practice, as well as whether one (art or practice) should incorporate the other's premises.

Gaps between the academy and the industry in the software and the IS fields are reportedly common (as in [115] and [48]) and it is prudent not to misjudge one based on any assumption of *what it should be*. As technology develops and changes fast [77, 31], and given that IS research has long followed, with a natural delay, developments made in the industry [16] and that practitioners do not use to read or consider academic research [93], the occurrence of such gaps is not surprisingly new. Indeed, the rigor-*versus*-relevance debate is familiar (e.g., in [4, 17, 32, 78, 74]). In particular, an arresting theme for empirical research is to investigate the very implementation of metrics in software process improvement initiatives (as identified in [62]), since metrics are essential for contrasting companies on process maturity [98].

In what comes to future theoretical research, the primary need is doubtlessly to consolidate terminology, principles and methods for measurement in software engineering, as denounced in [103] and [2], and subsequently continue to unify previous research outcomes by means of, for instance, meta-analyses. Our study tried to make bold developments in this sense. Even though subjectivity is still endemic to measurement (not only regarding its epistemological base, but also due to the generous current involvement with qualitative measures), one cannot ignore that it is of pressing need that the software community once for all agrees on the core assumptions. Besides the theoretical implications, one can easily devise productivity problems that may otherwise continue to happen.

Another theoretical issue that should be investigated in depth concerns the cultural components that are to influence or to be changed by the implementation of a metrics plan. Software teams already work in a performance-oriented way [94], but enforcing the adoption of, say, best practices in measurement is not to be without objections by the knowledge workers [105]. Moreover, organizations are regarded as routine-preserving structures [86, 116], maybe because it is harder to dissolve knowledge in order to learn something new (in this case, moving to a metrics culture) than to learn something for the very first time [59].

Finally, we agree that software engineering is engaged in a thorough, open investigation of its bases, and by means of merging with other knowledge areas — notably human resources management, marketing, organizational theories, and research methods — it is moving towards a more complete perspective on software development and consequences for the organizational information systems.

References

1. A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp (eds.), *Guide to the Software Engineering Body of Knowledge — Trial Version 1.00* (IEEE Computer Society Press, Los Alamitos, 2001).
2. A. Abran, A. Sellami, and W. Suryn, Metrology, measurement and metrics in software engineering, in *Proc. 9th Int. Software Metrics Symposium*, IEEE, Sydney, 3–5 September, 2003.

3. P. J. Ågerfalk and B. Fitzgerald, Flexible and distributed software processes: Old petunias in new bowls? *Commun. ACM* **49**(10) (2006) 27–34.
4. L. M. Applegate and J. L. King, Rigor and relevance: Careers on the line, *MIS Quarterly* **23**(1) (1999) 17–18.
5. D. E. Avison and G. Fitzgerald, Information systems development, in *Rethinking Management Information Systems: An Interdisciplinary Perspective*, eds. W. L. Currie and B. Galliers (Oxford University Press, New York, 1999), pp. 250–278.
6. E. Babbie, *Métodos de Pesquisa de Survey* (Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, 1999) [in Portuguese: "Survey Research Methods"].
7. R. P. Bagozzi, Y. Yi, and L. W. Phillips, Assessing construct validity in organizational research, *Administrative Science Quarterly* **36**(3) (1991) 421–458.
8. L. Bardin, *Análise de Conteúdo*, Edições 70 (Lisbon, 1977) [in Portuguese: "Content Analysis"].
9. M. O. Barros, C. M. L. Werner, and G. H. Travassos, Supporting risks in software project management, *J. Systems and Software* **70**(1–2) (2004) 21–35.
10. V. R. Basili, Is there a future for empirical software engineering? in *Proc. ISESE'06*, ACM, Rio de Janeiro, Brazil, 21–22 September, 2006.
11. V. R. Basili and H. D. Rombach, The TAME project: Towards improvement-oriented software environments, *IEEE Trans. Software Engineering* **14**(6) (1988) 758–773.
12. S. Beecham, T. Hall, C. Britton, M. Cottee, and A. Rainer, Using an expert panel to validate a requirements process improvement model, *J. Systems and Software* **76**(3) (2005) 251–275.
13. S. Beecham, T. Hall, and A. Rainer, Software process improvement problems in twelve software companies: An empirical analysis, *Empirical Software Engineering* **8**(1) (2003) 7–42.
14. C. G. P. Bellini, M.E.T.R.I.C.S. — Model for Eliciting Team Resources and Improving Competence Structures. A Socio-technical Treatise on Managing Customer Professionals in Software Projects for Enterprise Information Systems, Ph.D. dissertation, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2006.
15. C. G. P. Bellini, J. L. Becker, and D. Borenstein, Towards a better understanding of stakeholders' roles in software customization, *Int. J. Computers, Systems and Signals* **5**(1) (2004) 16–31.
16. I. Benbasat, D. K. Goldstein, and M. Mead, The case research strategy in studies of information systems, *MIS Quarterly* **11**(3) (1987) 369–386.
17. I. Benbasat and R. W. Zmud, Empirical research in information systems: The practice of relevance, *MIS Quarterly* **23**(1) (1999) 3–16.
18. J. H. Blackstone Jr., L. R. Gardiner, and S. C. Gardiner, A framework for the systemic control of organizations, *Int. J. Production Research* **35**(3) (1997) 597–609.
19. J. T. Boardman, Wholes and parts — A systems approach, *IEEE Trans. Systems, Man and Cybernetics* **25**(7) (1995) 1150–1161.
20. J. Bosch and L. Lundberg, Software architecture — Engineering quality attributes, *J. Systems and Software* **66**(3) (2003) 183–186.
21. M. C. Boudreau, D. Gefen, and D. W. Straub, Validation in information systems research: A state-of-the-art assessment, *MIS Quarterly* **25**(1) (2001) 1–16.
22. P. Bourque, R. Dupuis, A. Abran, J. W. Moore, and L. Tripp, The guide to the software engineering body of knowledge, *IEEE Software* **16**(6) (1999) 35–44.
23. L. C. Briand, K. El Emam, and S. Morasca, Theoretical and empirical validation of software product measures, Technical Report ISERN-95-03, 1995.
24. L. C. Briand, K. El Emam, and S. Morasca, On the application of measurement theory in software engineering, *Empirical Software Engineering* **1**(1) (1996) 61–88.

25. L. C. Briand, S. Morasca, and V. R. Basili, An operational process for goal-driven definition of measures, *IEEE Trans. Software Engineering* **28**(12) (2002) 1106–1125.
26. M. E. Bush and N. E. Fenton, Software measurement: A conceptual framework, *J. Systems and Software* **12**(3) (1990) 223–231.
27. J. K. Chhabra, K. K. Aggarwal, and Y. Singh, Code and data spatial complexity: Two important software understandability measures, *Information and Software Technology* **45**(8) (2003) 539–546.
28. G. A. Churchill Jr., A paradigm for developing better measures of marketing constructs, *J. Marketing Research* **16** (1979) 64–73.
29. D. S. Corbin, Establishing the software development environment, *J. Systems Management* **42**(9) (1991) 28–31.
30. W. L. Currie and B. Galliers (eds.), *Rethinking Management Information Systems: An Interdisciplinary Perspective* (Oxford University Press, New York, 1999).
31. W. L. Currie and I. A. Glover, Hybrid managers: An example of tunnel vision and regression in management research, in *Rethinking Management Information Systems: An Interdisciplinary Perspective*, eds. W. L. Currie and B. Galliers (Oxford University Press, New York, 1999), pp. 417–443.
32. T. H. Davenport and M. L. Markus, Rigor versus relevance revisited: Response to Benbasat and Zmud, *MIS Quarterly* **23**(1) (1999) 19–23.
33. C. Dekkers and P. McQuaid, The dangers of using measurement to (mis)manage: Measuring the software process, in *Proc. ASQ's Annual Quality Congress* (American Society for Quality, Milwaukee, 2002), pp. 551–560.
34. R. F. DeVellis, *Scale Development — Theory and Applications* (Sage, Newbury Park, 1991).
35. J. A. Díez, A hundred years of numbers. A historical introduction to measurement theory 1887–1990. Part I: The formation period. Two lines of research: Axiomatics and real morphisms, scales and invariance, *Studies in History and Philosophy Sciences* **28**(1) (1997) 167–185.
36. L. Donaldson, The normal science of structural contingency theory, in *Handbook of Organization Studies*, eds. S. R. Clegg, C. Hardy, and W. R. Nord (Sage, London, 1996), pp. 57–76.
37. L. M. Duvall, A study of software management: The state of practice in the United States and Japan, *J. Systems and Software* **31**(2) (1995) 109–124.
38. T. Dybå, Enabling software process improvement: An investigation of the importance of organizational issues, *Empirical Software Engineering* **7**(4) (2002) 387–390.
39. H. Eisner, *Essentials of Project and Systems Engineering Management* (John Wiley and Sons, New York, 1997).
40. K. El Emam, Software engineering process, in *Guide to the Software Engineering Body of Knowledge — Trial Version 1.00*, Chap. 9, eds. A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp (IEEE Computer Society Press, Los Alamitos, 2001), pp. 137–154.
41. W. M. Evanco and R. Lacovara, A model-based framework for the integration of software metrics, *J. Systems and Software* **26**(1) (1994) 77–86.
42. S. Faraj and L. Sproull, Coordinating expertise in software development teams, *Management Science* **46**(12) (2000) 1554–1568.
43. A. V. Feigenbaum, How to manage for quality in today's economy, *Quality Progress* **34**(5) (2001) 26–27.
44. N. E. Fenton and M. Neil, Software metrics: Successes, failures and new directions, *J. Systems and Software* **47**(2–3) (1999) 149–157.

45. N. E. Fenton and M. Neil, Software metrics: Roadmap, in *Proc. 22nd Int. Conf. on Software Engineering*, ACM, Limerick, 4–11 June, 2000, pp. 357–370.
46. N. V. Flor, Globally distributed software development and programming, *Commun. ACM* **49**(10) (2006) 57–58.
47. J. M. Ford, T. A. Stetz, M. M. Bott, and B. S. O'Leary, Automated content analysis of multiple-choice test item banks, *Social Science Computer Review* **18**(3) (2000) 258–271.
48. M. B. Franzen and C. G. P. Bellini, Arte ou prática em teste de software? *Revista Eletrônica de Administração* **11**(3) (2005) [in Portuguese: "Art or practice in software testing?"].
49. A. Fuggetta, L. Lavazza, S. Morasca, S. Cinti, G. Oldano, and E. Orazi, Applying GQM in an industrial software factory, *ACM Trans. Software Engineering and Methodology* **7**(4) (1998) 411–448.
50. R. A. Gardner, Resolving the process paradox, *Quality Progress* **34**(3) (2001) 51–59.
51. D. Gefen, D. W. Straub, and M.-C. Boudreau, Structural equation modeling and regression: Guidelines for research practice, *Commun. AIS* **4** (2000) 1–76.
52. C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering* (Prentice-Hall, Englewood Cliffs, 1991).
53. R. L. Glass, I. Vessey, and V. Ramesh, Research in software engineering: An analysis of the literature, *Information and Software Technology* **44**(8) (2002) 491–506.
54. A. Gopal, M. S. Krishnan, T. Mukhopadhyay, and D. R. Goldenson, Measurement programs in software development: Determinants of success, *IEEE Trans. Software Engineering* **28**(9) (2002) 863–875.
55. J. F. Hair Jr., R. E. Anderson, R. L. Tatham, and W. C. Black, *Multivariate Data Analysis* (Prentice Hall, Upper Saddle River, 1998).
56. N. Hanebutte, C. S. Taylor, and R. R. Dumke, Techniques of successful application of factor analysis in software measurement, *Empirical Software Engineering* **8**(1) (2003) 43–57.
57. W. Harrison, A flexible method for maintaining software metrics data: A universal metrics repository, *J. Systems and Software* **72**(2) (2004) 225–234.
58. W. Hetzel, The measurement process, in *Applying Software Metrics*, eds. P. Oman and S. L. Pfleeger (IEEE, Los Alamitos, 1997), pp. 72–93.
59. G. H. Hofstede, *Cultures and Organizations* (Harper Collins, London, 1994).
60. R. Hoving, Executive response: Project management process maturity as a "secret weapon", *MIS Quarterly Executive* **2**(1) (2003) 29–30.
61. J. Hussey and R. Hussey, *Business Research* (Palgrave Macmillan, New York, 1997).
62. J. Iversen and L. Mathiassen, Cultivation and engineering of a software metrics program, *Information Systems Journal* **13**(1) (2003) 3–19.
63. M. C. Jackson and P. Keys, Towards a system of systems methodologies, *J. Operational Research Society* **35**(6) (1984) 473–486.
64. M. Jørgensen, Software quality measurement, *Advances in Engineering Software* **30**(12) (1999) 907–912.
65. M. Jørgensen, A review of studies on expert estimation of software development effort, *J. Systems and Software* **70**(1–2) (2004) 37–60.
66. M. Jørgensen, K. H. Teigen, and K. Moløkken, Better sure than safe? Over-confidence in judgement-based software development effort prediction intervals, *J. Systems and Software* **70**(1–2) (2004) 79–93.
67. R. S. Kaplan and D. P. Norton, *The Balanced Scorecard* (Harvard Business School Press, Boston, 1996).

68. P. Katerattanakul and B. Han, Are European IS journals under-rated? An answer based on citation analysis, *European Journal of Information Systems* **12**(1) (2003) 60–71.
69. M. Keil, J. Mann, and A. Rai, Why software projects escalate: An empirical analysis and test of four theoretical models, *MIS Quarterly* **24**(4) (2000) 631–664.
70. T. M. Khoshgoftaar, E. B. Allen, and D. L. Lanning, An information theory-based approach to quantifying the contribution of a software metric, *J. Systems and Software* **36**(2) (1997) 103–113.
71. T. M. Khoshgoftaar and N. Seliya, Fault prediction modeling for software quality estimation: Comparing commonly used techniques, *Empirical Software Engineering* **8**(3) (2003) 255–283.
72. S. R. Kirk and S. Jenkins, Information theory-based software metrics and obfuscation, *J. Systems and Software* **72**(2) (2004) 179–186.
73. B. Kitchenham, Procedures for performing systematic reviews, Joint Technical Report — Keele University Technical Report TR/SE-0401 and National ICT Australia Ltd., Technical Report 040011T.1, 2004.
74. A. S. Lee, Rigor and relevance in MIS research: Beyond the approach of positivism alone, *MIS Quarterly* **23**(1) (1999) 29–34.
75. H. K. N. Leung, Quality metrics for intranet application, *Information and Management* **38**(3) (2001) 137–152.
76. M. Lindvall, R. T. Tvedt, and P. Costa, An empirically-based process for software architecture evaluation, *Empirical Software Engineering* **8**(1) (2003) 83–108.
77. F. Lopes and P. Morais, Lessons learned from the teaching of IS development, *J. Information Technology Education* **1**(2) (2002) 103–112.
78. K. Lyytinen, Empirical research in information systems: On the relevance of practice in thinking of IS research, *MIS Quarterly* **23**(1) (1999) 25–28.
79. S. G. MacDonell and A. R. Gray, Software engineering management, in *Guide to the Software Engineering Body of Knowledge — Trial Version 1.00*, Chap. 8, eds. A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp (IEEE Computer Society Press, Los Alamitos, 2001), pp. 121–135.
80. L. Mari, The meaning of "quantity" in measurement, *Measurement* **17**(2) (1996) 127–138.
81. L. Mari, Epistemology of measurement, *Measurement* **34**(1) (2003) 17–30.
82. J. G. McDaniel, Improving system quality through software evaluation, *Computers in Biology and Medicine* **32**(3) (2002) 127–140.
83. S. Morasca, Foundations of a weak measurement-theoretic approach to software measurement, in *Proc. 6th Int. Conf. on Fundamental Approaches to Software Engineering*, Warsaw, 7–11 April, 2003, pp. 200–215.
84. J. Münch and J. Heidrich, Software project control centers: Concepts and approaches, *J. Systems and Software* **70**(1–2) (2004) 3–19.
85. N. A. Mylonopoulos and V. Theoharakis, Global perceptions of IS journals, *Commun. ACM* **44**(9) (2001) 29–33.
86. R. Nelson and S. G. Winter, *An Evolutionary Theory of Economic Change* (Belknap Press, Cambridge, 1982).
87. P. Oman and S. L. Pfleeger (eds.), *Applying Software Metrics* (IEEE, Los Alamitos, 1997).
88. J. S. Osmundson, J. B. Michael, M. J. Machniak, and M. A. Grossman, Quality management metrics for software development, *Information and Management* **40**(8) (2003) 799–812.

89. P. Palvia, E. Mao, A. F. Salam, and K. S. Soliman, Management information systems research: What's there in a methodology? *Commun. AIS* **11** (2003) 289–309.
90. S. C. Palvia, R. S. Sharma, and D. W. Conrath, A socio-technical framework for quality assessment of computer information systems, *Industrial Management and Data Systems* **101**(5) (2001) 237–251.
91. D. Parnas, Agile methods and GSD: The wrong solution to an old but real problem, *Commun. ACM* **49**(10) (2006) 29.
92. R. Pavur, M. Jayakumar, and H. Clayton, Software testing metrics: Do they have merit? *Industrial Management and Data Systems* **99**(1) (1999) 5–10.
93. J. M. Pearson, A. Pearson, and J. P. Shim, The relevancy of information systems research: The practitioner's view, *Information Resources Management Journal* **18**(3) (2005) 50–67.
94. A. Peled, Creating winning information technology project teams in the public sector, *Team Performance Management* **6**(1/2) (2000) 6–14.
95. S. L. Pfleeger, Maturity, models, and goals: How to build a metrics plan, *J. Systems and Software* **31**(2) (1995) 143–155.
96. M. T. Pich, C. H. Loch, and A. De Meyer, On uncertainty, ambiguity, and complexity in project management, *Management Science* **48**(8) (2002) 1008–1023.
97. R. S. Pressman, *Software Engineering: A Practitioner's Approach* (McGraw-Hill, New York, 2001).
98. A. Rainer and T. Hall, A quantitative and qualitative analysis of factors affecting software processes, *J. Systems and Software* **66**(1) (2003) 7–21.
99. B. Ramesh, L. Cao, K. Mohan, and P. Xu, Can distributed software development be agile? *Commun. ACM* **49**(10) (2006) 41–46.
100. T. Ravichandran and A. Rai, Quality management in systems development: An organizational system perspective, *MIS Quarterly* **24**(3) (2000) 381–415.
101. L. D. Richards and S. K. Gupta, The systems approach in an information society: A reconsideration, *J. Operational Research Society* **36**(9) (1985) 833–843.
102. G. B. Rossi, A probabilistic model for measurement processes, *Measurement* **34**(2) (2003) 85–99.
103. F. Ruiz, M. Genero, F. García, M. Piattini, and C. Calero, A proposal of a software measurement ontology, in *Proc. 4th Argentine Symposium of Software Engineering*, SADIO, Buenos Aires, 1–3 September, 2003.
104. W. S. Sarle, Measurement theory: Frequently asked questions, in *Disseminations of the International Statistical Applications Institute*, ed. W. S. Sarle, ACG, Wichita, 1995, pp. 61–66.
105. H. Scarbrough, The management of knowledge workers, in *Rethinking Management Information Systems: An Interdisciplinary Perspective*, eds. W. L. Currie and B. Galliers (Oxford University Press, New York, 1999), pp. 474–496.
106. F. L. Schmidt and J. E. Hunter, Theory testing and measurement error, *Intelligence* **27**(3) (1999) 183–198.
107. A. H. Segars, Assessing the unidimensionality of measurement: A paradigm and illustration within the context of information systems research, *Omega* **25**(1) (1997) 107–121.
108. N. G. Shaw, Identifying relationships among factors in IS implementation, *Commun. AIS* **11** (2003) 155–165.
109. H. J. Smith and M. Keil, The reluctance to report bad news on troubled software projects: A theoretical model, *Information Systems Journal* **13**(1) (2003) 69–95.
110. I. Sommerville, *Software Engineering* (Addison-Wesley, Harlow, 2001).

111. I. Stamelos, L. Angelis, M. Morisio, E. Sakellaris, and G. L. Bleris, Estimating the development cost of custom software, *Information and Management* **40**(8) (2003) 729–741.
112. P. Stein, By their measures shall ye know them, *Quality Progress* **34**(5) (2001) 72–74.
113. D. W. Straub, Validating instruments in MIS research, *MIS Quarterly* **13**(2) (1989) 147–169.
114. D. W. Straub, D. L. Hoffman, B. W. Weber, and C. Steinfield, Toward new metrics for net-enhanced organizations, *Information Systems Research* **13**(3) (2002) 227–238.
115. P. A. Todd, J. D. Mckeen, and R. B. Gallupe, The evolution of IS job skills: A content analysis of IS job advertisements from 1970 to 1990, *MIS Quarterly* **19**(1) (1995) 1–27.
116. P. S. Tolbert and L. G. Zucker, The institutionalization of institutional theory, in *Handbook of Organization Studies*, eds. S. R. Clegg, C. Hardy, and W. R. Nord, Sage, London, 1996, pp. 175–190.
117. P. Wagstrom and J. Herbsleb, Dependency forecasting in the distributed agile organization, *Commun. ACM* **49**(10) (2006) 55–56.
118. D. Wallace and L. Reeker, Software quality, in *Guide to the Software Engineering Body of Knowledge — Trial Version 1.00*, Chap. 11, eds. A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp, IEEE Computer Society Press, Los Alamitos, 2001, pp. 165–183.
119. I. Wieczorek, Improved software cost estimation — a robust and interpretable modelling method and a comprehensive empirical investigation, *Empirical Software Engineering* **7**(2) (2002) 177–180.
120. C. Wohlin and A. A. Andrews, Prioritizing and assessing software project success factors and project characteristics using subjective data, *Empirical Software Engineering* **8**(3) (2003) 285–308.
121. M. H. Zack, Managing codified knowledge, *Sloan Management Review* **40**(4) (1999) 45–58.
122. K. Zhu and K. L. Kraemer, E-commerce metrics for Net-enhanced organizations: Assessing the value of e-commerce to firm performance in the manufacturing sector, *Information Systems Research* **13**(3) (2002) 275–295.