Abstract

Measurements serve as vital instruments to control projects involving software development outsourcing. However, managers have found it difficult to develop and implement effective measurement programs, in part because guidelines for choosing among concrete measurements are scarce. We address this gap between research and practice by examining frameworks and guidelines in the software process improvement literature. Our contribution comprises a framework that provides a set of measurements (selected from the research literature) for control of software development in a cooperative setting and a set of principles and guidelines for the design of an information infrastructure that provides managers with control information. We have validated our approach by showing our framework to an expert in outsourcing projects, who confirmed its potential.

Keywords

Software process improvement, measurements, metrics, outsourcing.

Acknowledgements

Thanks to the expert who confirmed our findings, helping us to validate the results presented in this paper. We are also grateful for the helpful comments we received from the reviewers of this paper.
1 Introduction

Making decisions that are better for value creation in current fast-changing software development scenarios demands dynamic monitoring and control mechanisms. Control through metrics is a critical component of the success of software process improvement programs (SPI) [14,18]. Much has been written about metric program implementation in SPI efforts [20,7,13,1,11,8]. But despite the importance of metrics, few studies have been made of its role in enabling new networked outsourcing models or of the software process by which control is improved through metrics. To address this gap, in this paper we aim at selecting metrics that help managers of outsourcing projects to make software that helps clients to meet their business goals in the current networked context of development processes.

To examine these issues, this explorative study was designed to address the following questions: a) how can managers increase control without losing flexibility? b) what has been written about concrete metrics – their implementation problems, benefits and contexts – managers can choose from? and c) what challenges did companies face when implementing software development metrics?

To answer these questions, a systematic literature survey has been carried out to develop a picture of the shape of the measurements field through the lens of software project improvement in an outsourcing context. By developing such a picture, we expose the ubiquity of approaches, help practitioners to synthesise and reflect on existing work and contribute to focusing the direction of interest of project managers. The current paper presents a subset of the results of this study in the form of a framework (Section 3) that contains a set of organisational effectiveness measurements and an information infrastructure that collects and distributes principles, lessons learnt and measurement data in software development projects. This framework is distilled from research findings published over the last two decades at the intersection of three different, but related fields: software process improvement (SPI), metrics, and outsourcing. An overview of the relevant parts of this field is presented in Section 2.

2 SPI, Metrics and Outsourcing

2.1 SPI

The fundamental objective of software process improvement (SPI) approaches is to “change software practices in order to achieve improvements in quality and productivity” [1]. Attempting to improve software quality, stakeholder satisfaction and profitability, SPI techniques address a number of topics such as software processes, standardisation, software metrics and project management.

Aaen et al. made a survey of the SPI literature and experiences from SPI practice [1]. The examples from practice described in their paper illustrate that there is room to implement SPI plans in very different ways and that metrics must be adapted, at the time of implementing them, to the specifics of an organisational environment.

All in all, although some of the approaches are extensively applied and offer unquestionable benefits such as the possibility to evaluate an organisation against stable criteria, systematisation and prioritisation [12], there is almost no discussion about how current networking conditions impact metrics on SPI programs. In particular, how do current global development conditions affect the implementation of metrics supporting an SPI initiative?

2.2 Metrics

People use measurements to gather feedback regarding the state of a project (for instance, completeness, quality of requirements, and accuracy of project estimations), create orientation during the change process, and determine how much the organisation has benefited from the changes derived from improvement plans. Visible results are considered critical to success of any improvement plan:
they keep participants focussed and motivated. In particular, it has been argued that members of an organisation would put more effort in SPI activities if measurements prove a return on investment [14].

However, measuring involves several risks. Organisations experience difficulties gathering and applying measurements that are meaningful [1]. Having numbers to show does not mean per-se neither that the measurements are relevant and meaningful, nor that they are accurate and reliable. Opportunistic behaviour, for instance, might jeopardise any SPI effort on the grounds of protecting particular interests.

Iversen and Mathiassen report from a case study that analyses an engineering process in which a metrics program is constructed and put into use [13]. The program’s goal was to test the effect of ongoing SPI initiatives within the organisation. This article is of interest to us because it describes a measurement program that went beyond the barrier of gathering data. Once implemented in the company Danske Data, the measurement program generated an “evolutionary cultivation process”. Moreover, Iversen and Kautz [15] and Kautz [17] emphasise that to be successful, the metrics programs implemented should be defined according to the organisation’s specific information needs.

All in all, measurements can be regarded as one of the means to gather feedback concerning the effect of the SPI effort, establishing baselines in the SPI programs and to demonstrating the extent to which the goals of a program are met. In spite of the clear importance of metrics, we have observed that outsourcing organisations are still lacking a portfolio of metrics defined to their specific information needs.

### 2.3 Outsourcing

Outsourcing today is mostly performed in an inter-organisational network rather than by a single organisation. This change in the context of software development makes managers face the challenge of having to control actions that are beyond traditional boundaries. Moreover, in this new context, open source and outsourcing development share challenges related to geographically distributed development such as project members working in arbitrary locations, rarely or never meeting face to face and coordinating activities exclusively via e-mails and bulletin boards.

Mockus, Fielding and Herbsleb examined the development process of an open source application by quantifying elements of software development such as developer participation, core team size, and code ownership for the Apache web server open source software development project [19]. The study shows that a large network of people (400 code contributors) cooperated to develop software and that most of the code was made by a small group of developers (approximately 15 developers). It was expected that these 15 developers arranged a partition of the code, to prevent making conflicting changes. But measurements proved otherwise: parts of the system requiring changes were worked upon by more than one developer, suggesting thus a healthy contribution coordination mechanism based on mutual trust and respect.

The practical nature of the coordinating contribution mechanism, however, remains to be researched. Specially viewing today’s highly competitive outsourcing market and recent research which has shown that not only product quality is important: low customer perception of delivery quality may rule out a supplier for the next project [6]. We hypothesise that part of this project success was due to the well covered open source system’s information needs; which was supported by the metrics they used.

### 3 Research findings

We present our research findings in a framework that consists of two parts: a set of organisational effectiveness measurements (Section 3.1) and a set of information infrastructure principles (Section 3.2). Organisational effectiveness measurements are software metrics in a broader context, as we explain below. The information infrastructure principles are the starting points for the design of a system of components that provides managers with information to control software processes.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Organisation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional size</td>
<td>Hewlett Packard, Eclipse, Danale Data</td>
<td>Gray66*, Inveren00</td>
</tr>
<tr>
<td>Developer participation: how is work distributed? E.g., can we recognise ‘partitioning’ in the code?</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Adherence to schedule: variation from agreed time of delivery, absolute and relative to volume of project</td>
<td>Danale Data</td>
<td>Inveren00, 03 and 06</td>
</tr>
<tr>
<td>Adherence to budget: variation from estimated size of resources</td>
<td>Danale Data</td>
<td>Inveren00, 03 and 06</td>
</tr>
<tr>
<td>Number of error reports relative to size in function points</td>
<td>Danale Data</td>
<td>Inveren00, 03 and 06</td>
</tr>
<tr>
<td>Defect density</td>
<td>AT&amp;T, Motorola, IBM, Apache server, Eclipse, Danale Data</td>
<td>Sannic00*, Reventl00*, Florin97, Mozilla00</td>
</tr>
<tr>
<td>Defects per line of documentation</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Defects per 10000 lines of code</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Poor release defects per thousand lines of code edited, instead of delivered.</td>
<td>Apache Server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Non-commented source statements per engineering month</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Cyclomatic Complexity</td>
<td>IBM Rochester SPI</td>
<td>Kard8* , McLaughlin10</td>
</tr>
<tr>
<td>System partitioning</td>
<td>IBM Rochester SPI</td>
<td>Kard8*</td>
</tr>
<tr>
<td>Fan In</td>
<td>IBM Rochester SPI</td>
<td>Kard8* , Florin97</td>
</tr>
<tr>
<td>Fan Out</td>
<td>IBM Rochester SPI</td>
<td>Kard8* , Florin97</td>
</tr>
<tr>
<td>Customer satisfaction</td>
<td>IBM Rochester SPI</td>
<td>Kard8*</td>
</tr>
<tr>
<td>Satisfaction with the development process (questionnaire)</td>
<td>Data Danale</td>
<td>Inveren00, 03, and 06</td>
</tr>
<tr>
<td>Employee satisfaction</td>
<td>IBM Rochester SPI</td>
<td>Kard8*</td>
</tr>
<tr>
<td>Satisfaction with the development process (questionnaire)</td>
<td>Data Danale</td>
<td>Inveren00, 03, and 06</td>
</tr>
<tr>
<td>Market performance</td>
<td>Not applicable or unreported</td>
<td>McLaughlin10</td>
</tr>
<tr>
<td>Loss of reputation of the firm</td>
<td>Not applicable or unreported</td>
<td>McLaughlin10</td>
</tr>
<tr>
<td>Satisfaction with the development process (questionnaire)</td>
<td>Data Danale</td>
<td>Inveren00, 03, and 06</td>
</tr>
<tr>
<td>Number of individuals submitting reports (e.g., bugs)</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Size of the development community</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Core team size</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Cumulative distribution of contributions to the code base</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Resources used to develop the system relative to volume of project (hours/SPF)</td>
<td>Data Danale</td>
<td>Inveren00, 03 and 06</td>
</tr>
<tr>
<td>Gain per Year in Productivity</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Gain per Year in Early Detection of Defects</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Average fixed-defects per working day</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Time used in review meetings</td>
<td>Small company</td>
<td>Inveren00</td>
</tr>
<tr>
<td>Reduction per Year in Calendar Time to Develop Software System</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Time to resolve problem reports</td>
<td>Apache server</td>
<td>Mozilla00</td>
</tr>
<tr>
<td>Percent overtime per 40 hours per week</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Resources used in coordination activities</td>
<td>Small company</td>
<td>Inveren00</td>
</tr>
<tr>
<td>Thousands of Dollars per Year Spent on SPI</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Dollars per Software Engineers per Year Spent on SPI</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Average engineering hours per fixed defect</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Reduction per Year in Post-Release Detect Reports</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Business Value Ratio of SPI Efforts</td>
<td>SEI Capability Maturity Model for Software</td>
<td>Herbied94</td>
</tr>
<tr>
<td>Average fixed-defects per working day</td>
<td>Hewlett Packard</td>
<td>Gray66*</td>
</tr>
<tr>
<td>Perception of delivery quality</td>
<td>Large consultancy company</td>
<td>VenExm08</td>
</tr>
</tbody>
</table>

*According to Rosa 1998

Fig. 1: The hierarchical diagram of measurements.

### 3.1 Organisational effectiveness measurements
Session I: will be adapted later by the editor

Principles

Start by determining goals
Define clear outcomes to expect and select the data based on clear objectives. An example of a clear outcome is to "improve delivery of software products every month". Using this basis for research will help in the program, being based on the goal that evaluations expect to achieve from the metrics program.

What not to do
To implement a set of metrics that is not well suited to describe a concrete outcome. An example of a too general objective is "to give information about the effect of improvement initiatives".

Quote/Example
"Implementing metrics for software development that can show how improvements are achieved is a great objective. However, trying to measure the return on investment for software development is an even greater objective. It requires a more detailed understanding of the processes and practices involved." - Cowan, 2000

Reference
Cowan00

Match your goals with meaningful attributes
To have a clear goal such as improve efficiency by 10% is important, but it does not necessarily mean that the chosen attributes to measure (and the metrics results) describe the programs goal.

What not to do
To undermine the importance of having a clear goal and pretend that attributes that describe the goal, leaving the question of what should be measured to the group responsible for the metrics program.

Quote/Example
"We expect to gain a 10% improvement in efficiency because the business has been an important focal point of the SPI effort. Major messages have already been delivered to the CEO not the line manager will be responsible for showing this 10% efficiency improvement." - Taylor, 2000

Reference
Taylor00

Establish incentive structures
Metrics programs seem to be more successful if people see that their efforts bring improvement to the process or the product. It should be clear:
1. what data to report,
2. how to report that data,
3. how results are reported.

What to do
To inform project managers about exactly what data they should collect, how they will have to report it, what part of the process and product will be better by analyzing that data.

Quote/Example
"Those who report data to a metrics program need to see some form of advantage in the program." - Spivey, 2000

Reference
Spivey00

Establish project
A metrics program should be a stand-alone initiative and therefore it is set in the context of a project for its own sake. It should make the task of collecting and analyzing the data easier. Moreover, you need the right staffing to carry on a measurement program.

What to do
To have people work on metrics as an extra task of their current projects, forgetting to recognize that collecting and reporting metrics consume resources (e.g., time).

Quote/Example
"Establishing a formal project made the program more viable in the organization and made it much easier for the participants to argue that adequate resources should be allowed." - Groves, 2000

Reference
Groves00

Short simple
In the beginning, collect a small set of goal-oriented metrics. For example, you could measure the number of fault change requests delivered on time and the time used in review meetings and found the change requests delivery on time had increased from 40% to 77% and review meeting time was shortened by a factor of 4.

What not to do
Neither to start with a large metrics set, nor to start with too general metrics. Too many resources will be spent on them and people will not see accordingly advantages when the large set of results comes back to them.

Quote/Example
"An early simple metrics should be measured, and all projects were required to report data from day one. This was an extremely ambitious undertaking, and as an early result of the efforts have not been measured, and some have even been officially abandoned." - Spivey, 2000

Reference
Spivey00

Plan to throw one away
Some measurements will prove to be too difficult to get right, despite initial assumptions and inactivity in the data access.

What to do
To underestimate the measurement. For instance, planning that the measurement will be made in a completely automatic way.

Quote/Example
"The first measurement report only included 20% of the projects and only three out of six factors." - Spivey, 2000

Reference
Spivey00

Use organizational knowledge
Organizational knowledge metrics programs must take into account the existing work practices in the organization, and the needs of the stakeholders affected by a potential effect of the metrics application.

What not to do
To not communicate clearly the advantages expected from implementing the program.

Quote/Example
"External consultants acted as analyzers of the current practice and carried out interviews with the developers. They provided the knowledge necessary to define metrics and to gather data." - Groves, 2000

Reference
Groves00

Consider potential problems when measuring your organization's goal
Measure size accurately is critical because size is a key attribute to measure common goals such as efficiency and productivity.

What to do
To believe literally the first result obtained from counting function points without checking that results match goal and match questions, i.e., checking that numbers obtained match the perception of the system.

Quote/Example
"Excluding a size measure varies greatly in its objective of measuring efficiency and productivity, as there was no longer a measure of the output of the software projects." - Groves, 2000

Reference
Groves00

Match measurement with your organization's goal
Attributes to measure, measurement data and its results need to be recognized by management. Measurement must describe part of the project in the eyes of stakeholders. Otherwise measurements become unacceptable.

What to do
To use measurements as just gathering data without realizing them to a business goal. For instance, to use function points count without normalizing them when needed.

Quote/Example
"After counting function points in several application systems, it was very difficult to see any relationship between the perceived complexity of the systems and the number that the counting procedure had arrived at." - Groves, 2000

Reference
Groves00

Here a complementary suite of measurements
The measurements you choose should be complementary. Each measurement should contribute to improve the picture of the system obtained from the measurement program.

What not to do
For instance, to count code performed during the original development twice.

Quote/Example
"Enhancement projects that continued work on an existing system were accredited the extra function point count of each module they modified even if the modifications were trivial. This gave the team the false perception of productivity, or in other words, unrealistically high productivity." - Spivey, 2000

Reference
Spivey00

Use implemented knowlege
Implementing metrics programs involves several phases of knowledge such as SPI, software development and reverse engineering. One solution might be to include external consultants.

What not to do
Organize unfamiliar areas of knowledge (for instance, reverse engineering if implementation has background is software architecture) when making decisions about how to implement the metrics program.

Quote/Example
"The first measurement report... was criticized for being too academic." - Groves, 2000

Reference
Groves00

Use non-invasive measurement whenever possible
Facilitate collecting data by making it simple. Metrics from finished projects would be used as a baseline and metrics of finished parts of projects (when collecting them makes no harm) should be used when possible. Beware that some stakeholders will not provide data.

What not to do
To have tedious mechanisms to report the data, especially with unclear questions.

Quote/Example
"There are some who simply do not enter data into the system. There are some that have misunderstood the definition of the field... results from <13 out of 50 projects that were completed>" - Groves, 2000

Reference
Groves00

Publish objectives and collected data widely
People need to see that metrics that they collected are used and that some advantage. Publishing available objectives is a way to secure gathering data of the quality for the SPI program, and to improve the metrics program with employer's feedback.

What not to do
Not only communicate the metrics, but mainly communicate the principles to discuss the metrics design and the results. Use their feedback to improve the measurement program.

Quote/Example
"The metrics program may result in undesirable outcomes. Another problem was that questionnaires / journal questionnaires relating to contractual issues, project status or the entire course of the project, whereas those who answered the questionnaires were users who were only involved in supervision tasks.

Reference
Koster00

Facilitate debate
To not hear what employees have to say about the metrics program and its implementation. This might cause the loss of valuable improvements for the program (such as improving the fraud fields of the system used to report data).

Quote/Example
"Another problem was that questionnaires / journal questionnaires relating to contractual issues, project status or the entire course of the project, whereas those who answered the questionnaires were users who were only involved in supervision tasks.

Reference
Koster00

Fig. 2: Information infrastructure: principles for the design of a system that provides managers with information to control software processes (part 1). This section reports software development outsourcing measurements for SPI found in the existing
Why do we use the term “organisational effectiveness measurements” instead of software metrics? As we have argued before, software metrics need to be applied in their organisational context, which, in current practice, often means globally distributed cooperative software development. The notion of organisational effectiveness as proposed by Applegate [2], which we explain below, provides us with the means to systematically identify the organisational context for software metrics. Our framework provides a collection of software (mostly process) metrics that instantiates Applegate’s notion for the domain of SPI.

The software metrics that comprise our set of organisational effectiveness measurements are metrics that help managers to control software development projects. We are specifically interested in metrics that project managers can use to control software development performed in an outsourcing project. Control in this context is the ability to develop an understanding of what is going on in the project online and make informed decisions. In particular, we want to understand how these measures can inform software development managers. This coincides exactly with Applegate’s notion of organisational effectiveness, which “concerns what to measure to provide information upon which to base management decisions.”[14]. The four areas of interest in measuring organisational effectiveness measurements adapted from Applegate [2], when instantiated with metrics we found in the SPI literature, are (i) results, which are needed to know how the software quality assurance process is performing, (ii) stakeholder satisfaction, (iii) industry dynamics, and (iv) software process performance, the set of “activities, methods and transformations that people use to develop and maintain software and the associated products, for example: product plans, design documents, code, test cases and user manuals” (SEI)).

Our selection of metrics is presented in Figure 1. Note that already in 1999, more than 487 metrics [4,10,9,16,22,23,5] for software process improvement had been identified [21]. Our selection comprises metrics that, according to existing literature, have been tried in real projects of real organisations, as is indicated in the column labelled ‘Organisations’. (The name ‘SEI Capability Maturity Model for Software” refers to organisations that have implemented that model.) This choice complies with our acceptance criterion: managers of outsourcing projects should find them useful for their software development outsourcing projects.

### 3.2 Information infrastructure

As stated before, we are interested in metrics that project managers can use to control software development performed in outsourcing. According to general models of control, a controlling system (in this case: a project manager) needs information about the system that it tries to control (in this case: a software development project in a cooperative context). The software metrics literature discusses the many different metrics identified in the software field that can serve as control information, and that we have presented in the previous section. This information, however, needs to be made available to the controlling system. The information infrastructure is the system that connects the controlled system to the controlling system and supplies the controlling system with information.

In Figure 2, we present a set of principles that can serve as a starting point for the design of such an information infrastructure. Like the set of metrics presented in the previous section, this set of principles is a selection of principles found in the research literature. We have selected these principles that, based on experiences of applying SPI programs, provide managers with information to control soft-
ware processes. Thus, the principles are taken from related work describing actual experiences in SPI measurements program application. We illustrate how our principles can be used with the following examples. Consider a manager of an outsourcing project who wants to have more control. A significant part of having that control is to know exactly who to contact in the off-shore team, which can be seen by measuring developer participation (i.e., how is work distributed? Is there any partitioning in the code? Our framework points out which metrics can be used for that; see the Organisational effectiveness metrics, Results, Code ownership, Developer participation in Figure 1).

As another example, consider a manager who has to implement, for the first time, SPI in his project and who needs to do so with very limited resources. He plans to base size measurement on function points and, in order to reduce the burden of extra work for his people, plans to compute function points automatically. Our framework shows that functional size is a measurement that has been used many times (Figure 1 under Results, size, functional size), and that it is critical (Figure 2 under Information Management, consider potential problems when measuring size). It also points out an example where after using function points, managers could not see any relationship between the perceived complexity of the system and the obtained metric results (Figure 2, Information management, match measurement with your organisation goals). This challenge could be explained by a known problem with function points: maintenance projects being credited the entire function point count of each module they modify even for tiny modifications, thus, indicating unrealistically high productivity. This reference is pointed at by our framework. In this way, using our framework, the manager is warned of the risk of automatizing function point counting.

3.3 Using the framework to derive guidelines for cooperation in outsourcing

In this section, to show usefulness of our work, we present a number of guidelines for software development in an outsourcing context. We view these guidelines as new insights that we were able to find by comparing traditional industrial styles of development with cooperative development processes from the perspective of the sets of metrics and principles provided by the framework.

**Distribution of work** Measure how work is distributed within the project (for instance, employee’s participation). In off-shored outsourcing projects there is most likely little if any control on work distribution within part of the team, distance being a large obstacle to exercise control in this way. However, measuring the distribution of work enables managers to improve control in the development process.

**Coordination mechanisms** Consider improving traditional coordination mechanisms such as plans, system-level design, and defined processes. We have seen cases of successful development projects where non-traditional mechanisms are used, such as social networking-like notifications of commit information in the development of the Apache server. These mechanisms have the potential to help managers save valuable resources.

**Feedback and frontier building** Respect the position of those with experience in the area being worked on. Respecting the frontiers built on the ground of experience supports beneficial feedback.

**Cohesion and coupling** Analyse coupling and cohesion to improve development processes. By cohesion we mean the number of intra-unit activities that generate knowledge, such as the amount of time spent in introducing new members to project tasks. By coupling we mean work that crosses the boundaries of a working group. For instance, coupling increases with the interactions between working units. Consider how many resources are invested in tasks related to cohesion and coupling: measuring coupling and cohesion have the potential to paint a picture of where resources go in a project.

3.4 Evaluation of the proposed framework

We have validated our framework via two interviews with an expert in managing outsourcing project. Reflecting on the potential of the framework applied to general situations, our expert could relate to this approach, finding it useful to improve control in outsourcing projects. In particular, the interviewee was very enthusiastic about the guidelines and principles, recognising specially the principle “Plan to through one away” and strongly agreeing with the guideline that suggest to analyse cohesion and
coupling.

Is our framework suited to address the outsourcing measurement challenges? The expert agreed that our framework would be useful in the current outsourcing context because it helps companies to meet the current outsourcing challenges such as lack of time, need for semi-automatization and competitiveness. In the opinion of this expert, our framework “is OK because metrics are important, but there is no time to report findings or searching for the best metrics: it is all about finishing a project on time. We would need some kind of automated system, so that there are fewer things to do.”

Surprisingly, our interviewee was concerned about the framework’s potential to replace managers. This was never the authors’ intention, who consider the thought of replacing people by frameworks totally unrealistic. The idea behind our framework is to empower managers by giving the right information.

For reasons of space, unfortunately we can neither detail our rigorous literature review and the metrics inclusion and exclusion criteria, nor compare this paper’s contribution with other existing methods in literature such as Basili’s Goal Question Metric method. However, we are aware of the importance of being rigorous in these regards and we are reporting these essential points in coming articles.

4 Discussion

This study addresses the question of what to measure to provide information upon which to base management decisions in software process improvement. We have based our study on research papers that report real experiences with metrics in organisations such as Rico’s a survey of almost 500 metrics [21]. Our framework is different from Rico’s in two ways. First, while Rico’s aim was to be complete, our aim is to provide a small set of metrics that (i) have proven to be useful in organisations and (ii) focus on cooperative software development, specifically outsourcing. Second, our selection contains a number of metrics that have been identified after the year in which Rico’s selection was published.

The results of our research suggest that organisations should develop mechanisms so that outsourcing companies can develop their own benchmarking standard to compare across the different kinds of projects in their organisation. Our work is, thus, in line with recent work in the area of software maintainability [3]. Our results suggest that the framework of Applegate has the potential to help us to find the right metrics. In that sense, our results are in line with Iversen and Ngwenyama [14], who, based on a longitudinal study of a change initiative in a Danish software company, tried to develop an understanding of measurements in SPI. Their organisational-change-theory-based measurement methodology can be considered an ancestor of our outsourcing measurement framework.

5 Conclusion

Despite an abundance of research literature on metrics for software process improvement, it is still difficult for managers to choose a set of measurements that enables them to control software development, especially for software development in a cooperative setting such as outsourcing. In this paper, based on a systematic survey of the research literature, we present a framework that provides a set of measurements and a set of principles and guidelines for the design of an information infrastructure. The set of measurements has been distilled from the research literature by selecting metrics that are well-defined, have been used in real projects in real organisations (as reported in the literature), and that we believe are most suitable for cooperative software development. The framework has been validated via expert interviews. The main conclusion is that the set of measurements has the potential to improve control in outsourcing projects, and that the principles and guidelines are potentially very useful for managers to apply the measurements in real-world projects, particularly in an outsourcing context. However, given the time pressure that is prevalent in today’s outsourcing market, it is important that any SPI project strives to automate as much measurements and reporting as possible.
6 Literature


7 Author CVs

Laura Ponisio

Laura received her Ph.D. in the area of software reverse engineering from the University of Bern (Switzerland), her Master in object-oriented software engineering from both Ecole des Mines de Nantes (France) and Vrije Universiteit Brussel (Belgium), and her Licentiate degree in Computer Science from UNLP, the National University of La Plata (Argentina). She has work experience in industry and academia, including software systems development and database administration. Currently, she performs action research in The Netherlands focusing on international collaboration with private and public institutions. Her research focuses on inter-organisational networks, global software development and object-oriented systems.

Pascal van Eck

Pascal van Eck received his MSc degree in Computer Science from Vrije Universiteit Amsterdam in 1995. From 1995 until January 2000, he worked as a research assistant at the Artificial Intelligence Department of Vrije Universiteit. In 2001, he successfully defended his PhD thesis on a formal, compositional semantic structure for the dynamics of multi-agent systems. Starting February, 2000, he works at the Information Systems Group of the Department of Computer Science, University of Twente, as an assistant professor. His research interests include IT governance and alignment of business and IT architectures at the enterprise level.