

# Measuring IT Infrastructure Project Size: Infrastructure Effort Points

Joost Schalken<sup>1</sup>, Sjaak Brinkkemper<sup>2</sup>, and Hans van Vliet<sup>1</sup>

<sup>1</sup> Vrije Universiteit, Department of Computer Science,  
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands  
{jpp.schalken, jc.van.vliet}@few.vu.nl

<sup>2</sup> Utrecht University, Institute of Information and Computing Sciences,  
Padualaan 14, 3584 CH Utrecht, The Netherlands  
{s.brinkkemper}@cs.uu.nl

**Abstract.** Our objective is to design a metric that can be used to measure the size of projects that install and configure COTS stand-alone software, firmware and hardware components. We call these IT infrastructure, as these components often form the foundation of the information system that is built on top of it. At the moment no accepted size metric exists for the installation and configuration of stand-alone software, firmware and hardware components. The proposed metric promises to be a viable instrument to assess the effectiveness and efficiency of IT infrastructure projects.

## 1 Introduction

Organizations no longer create software intensive systems from scratch. The use of pre-existing software components, not created by the organizations themselves, becomes ever more prevalent in the creation of large software systems [1, 2]. These pre-existing components are often called *commercial-of-the-shelf components* (COTS components) in software engineering literature. In this paper we however prefer to use the term *non-developmental items* (NDIs) [3] for these pre-existing components.

The integration of NDI components that are packaged as stand-alone programs differs significantly from traditional software development. Many software engineering metrics that have been applied to software development (such as function points [4, 5], lines of code [6], object points [7], or bang metrics [8]) cannot be applied to projects that integrate these NDI components into larger systems. After all most effort in infrastructural IT projects is not in programming a system, but in installing and configuring the system. In this paper we propose a new software metric to measure the size of projects that integrate stand-alone NDI components into software-intensive systems.

The metric we propose is not only applicable to the integration of stand-alone non-developmental software components, but also to the integration of

firmware<sup>1</sup> and hardware components into software-intensive systems. We use the term *Information Technology infrastructure* (IT infrastructure) to refer to NDI hardware, firmware and stand-alone software components, as these components often form the foundation of the information system that is built on top of it.

Examples of IT infrastructure projects are: operating system upgrades, installations of a database system, deployment of new desktop computers and memory upgrades of servers.

Up until now no size metric in the area of IT infrastructure development has received broad acceptance by industry. Although no standard size metric for IT infrastructure exists, it does not mean that there is no need for such a metric. On the contrary, the need for such a size metric is increasing. Empirical knowledge of NDI-based systems is still at an early stage of maturity [9]. Processes are quite different from traditional projects [10] and project estimation and tracking are less effective for NDI-based development [10]. This is problematic as the use of NDI components is becoming ever more prevalent [1, 2]. The metric we propose in this paper uses information part of which only becomes available late in a project. Consequently, it's intended use is to assess and validate the effectiveness and efficiency of projects, rather than upfront cost estimation.

### 1.1 IT Infrastructure Defined

The absence of consensus on the meaning of COTS within the academic community [3, 11] necessitates a definition of the related concept IT infrastructure in this section. The definition of IT infrastructure marks which projects can and which projects cannot be measured with the new size metric.

The term IT infrastructure has been inspired by the term technical infrastructure [12]. In this paper the following definition of *IT infrastructure development* will be used: “*the deployment, installation, connection and configuration of both new and upgraded, non-developmental hardware, firmware and stand-alone software components*”.

The development of IT infrastructure is concerned with pre-existing hardware and software components that have not been developed by the organizational unit that installs these components. In software engineering literature a pre-existing software component is often called a *commercial-of-the-shelf component* (COTS component). In this paper we however prefer to use the term *non-developmental item* (NDI) [3], as the term COTS implies that the component comes from a commercial vendor. In this paper the term NDI refers not only to software components, but also to firmware and hardware components.

Non-developmental software components can be packaged in several ways [11], either as source code, static libraries, dynamic libraries, binary components or stand-alone programs. The type of packaging also has a direct impact on how these components are integrated and delivered to the customer. NDI components

---

<sup>1</sup> As firmware has remarkable similarity with software (with respect to its configuration), everywhere where software is written, one should read software/firmware unless explicitly stated otherwise.

that are provided in source code or library form usually require programming to integrate the components into the software system under construction. These kinds of NDI components are usually delivered as inseparable subcomponents of the system under construction. On the other hand NDI components that are provided as stand-alone programs usually require little or no programming, but require so much the more configuration. These NDI components are usually not delivered as an inseparable system, but instead the components need to be installed separately or are installed by the installation program as separate, stand-alone components or programs.

The development of IT infrastructure not only entails the installation and configuration of stand-alone software components, but also the deployment, connection and configuration of hardware components. As the choice of software components is not independent hardware components and because the integration of the components is frequently performed in a single project, we have chosen to measure the size of software, firmware and hardware components using a single, composite size metric.

The development of IT infrastructure consists of *deploying* the hardware (placing the physical components), *installing* the stand-alone software (loading the software from the installation medium into the target hardware), *connecting* the hardware (installing sub-modules and wiring the hardware) and *configuring* the software (setting and testing the configurable parameters and settings).

## 1.2 Related Work

This section describes related work in the field of IT infrastructure size metrics and algorithmic cost estimation models for IT infrastructure. Cost estimation models are included in the overview, as they can be seen as size models that do not only take the inherent problem size into account, but also the capability of the environment to deal with the complexity at hand. In this article we focus solely on the costs of labor for the installation and configuration of the IT infrastructure. Selection effort, hardware costs, such as discussed in [13], and license costs lie outside the scope of this paper.

There are two cost estimation methods that are commonly used in practice: *IT infrastructure service costing* and *IT infrastructure product costing*. The IT infrastructure component is either viewed as a service whose costs are of a recurring nature or as a component that is delivered to an organization as product or a one-off service delivery. Examples of IT infrastructure services are cpu cycles on a mainframe and network ports. Examples of IT infrastructure products are configured servers and software upgrades.

The most crude approach to IT infrastructure costing consists of amortizing the total costs generated by a class of IT services or IT products by total amount of service units or products delivered.

A more sophisticated approach to IT infrastructure service costing is offered by Van Gorkom [14] in the form of Service Point Analysis. The Service Level Agreement is decomposed into Service Level Agreement components. Based on

the Service Level Agreement components standardized cost estimates can be made.

Another more sophisticated approach to IT infrastructure product sizing is the SYSPPOINT method [15]. The IT infrastructure product to be provided is divided into primitive components (servers, workstations, printers, LAN's, WAN's, server applications and client applications). Based on the relative complexity and count of the primitive components, tables can be used to calculate the total size of a project in SYSPPOINTS.

COCOTS [16, 17] is an advanced cost-estimation model for NDI software, based on the COCOMO suite of cost estimation models. The COCOTS model allows the estimation of not only the implementation of the system, but also the selection costs and modification costs. The COCOTS tailoring model estimates the implementation and configuration costs of a system, based on parameter specification, script writing, reports & gui, security and the availability of tailoring tools. Each of the factors is measured on a five-point scale.

The last method discussed in this section is data envelopment analysis, which can be applied to measure the relative efficiency in creating IT infrastructure [12]. Data envelopment analysis allows the efficiency of projects to be compared on a variety of output measures simultaneously. In that sense it is not a cost estimation or size measurement procedure, but the method does shine a light on the project's productivity. The method solves the problem that IT infrastructure can have multiple outputs (e.g. servers can have connected users, storage space and processing speeds as output measures).

Amortizing product or service costs, Service Point Analysis and the SYSPPOINT method share the drawback that estimates can only be made for IT infrastructural systems that consist of known infrastructural product or service components. The costs for IT infrastructural systems that contain novel product components or service components cannot be estimated.

Data envelopment analysis can only analyze the implementation efficiency of projects that implement similar products or services. Different IT infrastructural projects will yield very different primitive outputs. Compare the efficiency in providing network throughput with the efficiency of providing storage space (both measured in gigabytes). This explains why comparisons are only possible between projects that have similar end results, comparing throughput with storage of data is of course not meaningful.

Our method measures the size of IT infrastructure on a continuous, interval scale. It is reasonably precise, whereas COCOTS measures each attribute on a rough 5-point scale. The metric allows different types of IT infrastructural products to be compared to each other and does not depend on the existence of a list of known IT infrastructure components.

### 1.3 Structure of Paper

Having discussed the necessity of IT infrastructure metrics and the definition of IT infrastructure, the remainder of this paper is structured as follows: In Sect. 2 the metaphor that guided the design of the infrastructure metric is presented

together with the formal definition of the metric. Section 3 discusses the calibration of the measurement formulas presented in Sect. 2. Section 4 provides the results of the preliminary calibration and validation of the metric during the feasibility study. Section 5 describes the conceptual validation of the proposed metric for IT infrastructure. And the last section wraps up the paper with some concluding remarks and further work.

## 2 Infrastructure Effort Points

In this section we present our metric to measure the size of IT infrastructure projects, the *Infrastructure Effort Point* or IEP for short. First the the underlying principles that guided the design of the size metric are explained. Following the theory design of the metric a detailed description of Infrastructure Effort Points and its measurement procedure is given.

Infrastructure Effort Points only measure the size of the installation and configuration effort of the IT infrastructure. The NDI component selection effort, training effort, hardware costs, and license costs cannot be measured using this metric.

### 2.1 Theory Design

In this section we explain the design of the size metric for IT infrastructure using a metaphor. A metaphor helps to create a common understanding [18]. It can be used to explain the underlying principles and assumptions of a measurement procedure.

The metaphor gives an intuitive justification for a metric. For example function point analysis counts the variety and complexity of the data records that are received, stored, transmitted and presented by an information system. Function point analysis is able to measure the size of different classes of information systems by abstracting each information system to a manipulator of flows of information. This is the guiding metaphor of function point analysis. It is based on the assumption that the complexity of an information system is equal to the complexity of its information flows.

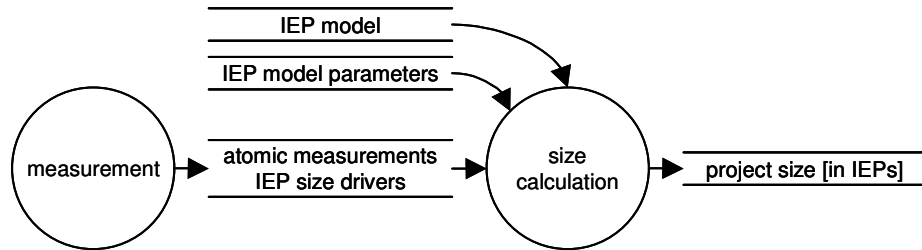


Fig. 1. Calculation of size based on project characteristics.

**Table 1.** Atomic measurements for IEP of hardware installation.

<i>Group</i>	<i>Size driver</i>	<i>Symbol</i>	<i>Unit of measurement</i>
main components	number of components	$c_i$	-
	average weight	$c_i^w$	kilo
	installation or removal	$c_i^a$	{installed, removed}
subcomponents	number of subcomponents	$s_{i,j}$	-
	average number of connections	$s_{i,j}^c$	-
	installation or removal	$s_{i,j}^a$	{installed, removed}
external connections	number of connections	$w_{i,j}$	-
	average length	$w_{i,j}^l$	meter
	installation or removal	$w_{i,j}^a$	{installed, removed}

Infrastructure Effort Point analysis considers the development of IT infrastructure to consist of two activities: wiring and placing hardware boxes during the deployment and connection of hardware components and the manipulation of configuration settings during the configuration of software components and configurable hardware.

The guiding metaphor for Infrastructure Effort Points is based on the following three assumptions:

1. Infrastructural IT projects are composed of a hardware and a software component.
2. The effort of the hardware component of the project depends on the number of hardware boxes that need to be installed and the number of connections that need to be made.
3. The effort of the software component of the project depends on the number of configuration parameters that need to be set.

## 2.2 Hardware Effort Points

Two distinct tasks in IT infrastructure projects have been identified: tasks related to hardware and tasks related to software. Verner and Tate [19] argue that different types of system components can have different size equations. In this section we identify the size drivers and size equations that are applicable to the deployment and connection of hardware components.

A bottom-up size model, as the Infrastructure Effort Points, consists of a number of size drivers and one or more size equations. A size driver is “any countable, measurable, or assessable element, construct, or factor thought to be related to the size” of a component [19]. The size drivers form the input to a size equation that combines the different counts on a specific size driver into a single size measurement.

For the hardware side of the Infrastructure Effort Point equation we distinguish three major size drivers: main components, subcomponents and connections. Each of the major size drivers has associated minor size drivers that can be

used to fine-tune the size equations in the future. Although technically there is no difference between a major and a minor size driver, practically we expect the major size drivers to have greater influence on the size of the IT infrastructure project.

The first major size driver is the number of main components  $c_i$  of a certain type of hardware that has been installed or removed. Main components are those pieces of hardware that are considered to form a functional whole by their average end users. E.g. an end user will identify a scanner as a main component, whereas the separate automatic document feeder for the scanner is seen as a subcomponent, as the automatic document feeder cannot be seen as an independent machine that offers functionality on its own. Associated minor size drivers are the average weight of the main component  $c_i^w$  and whether the components were installed or removed  $c_i^a$ .

The second major size driver is the number of subcomponents  $s_{i j}$  of a certain type that have been installed or removed from main component  $c_i$ . (The index  $i$  refers to the main component to which the subcomponents are attached, the index  $j$  refers to this particular group of subcomponents.) Subcomponents that have not been installed or removed, but instead were already assembled with the main component should not be counted. Minor size drivers that are associated with the size driver number of subcomponents are the average number of connections between the subcomponent and the main component and other subcomponents  $s_{i j}^c$  and whether the subcomponents were installed or removed  $s_{i j}^a$ .

The third and last major size driver for the hardware side is the number of connections  $w_{i j}$  between the outside world and main component  $c_i$ . The connections size driver considers all physical connections (wires) between the main component and the outside world, but not the mutual connections between subcomponents and connections between subcomponents and the main component as these have already been counted (in  $s_{i j}^c$ ). Examples of connections are the power cable and the network cable of a personal computer, but not the keyboard cord. Associated minor size drivers are the average length of the connection  $w_{i j}^l$  and whether the connections were installed or removed  $w_{i j}^a$ .

These three major size drivers and their associated minor size drivers form the input for the size equation that combines the measurements on the individual size drivers, see Fig. 1 for a schematic overview. The size equations consist of a model combined with calibrated model parameters. The equation model states which size drivers need to be taken into account and in which manner. E.g. a size model might state that the size of a task is equal to the number of main hardware components multiplied by a constant plus the number of connections multiplied by a constant, i.e. size  $s_i^{hw} = \theta_1 \cdot c_i + \theta_2 \cdot w_{i j}$ . The exact size equation is determined during the calibration process in which the most appropriate values for the constants in the equation are determined using empirical data.

The calibration of a size model, once the empirical data has been collected, is conceptually easy, albeit computer-intensive (more information on the calibration process can be found in Sect. 3). The selection of the appropriate model

**Table 2.** Atomic measurements for IEP of software configuration.

<i>Attribute</i>	<i>Metric</i>	<i>Symbol</i>	<i>Unit of measurement</i>
configuration parameters	number of parameters	$p_i$	-
	parameter type	$p_i^v$	{text, number, boolean, binary}
	input type	$p_i^t$	{gui, text interface, configuration file, configuration database, script file, dip-switch/jumper, other}
configuration group	number of parameters	$g_{i,j}$	-
	parameter type	$g_{i,j}^v$	<i>see above.</i>
	input type	$g_{i,j}^t$	<i>see above.</i>

is more complicated. Apart from the selection of an appropriate form for the equation (linear, logarithmic or quadratic) one needs to be careful to select the right amount of size drivers for the model. Too few size drivers makes the size equation perform poorly, as insufficient information is taken into account. Too many size drivers creates the risk of over-fitting the size model, causing the size equation not to capture the real hardware size but instead some random patterns that exist in the observed data. The risk of over-fitting the model is increased when many size drivers are included relative to the amount of empirical data.

For the first empirical validation we propose to use only a single size driver to prevent over-fitting the data. When more data becomes available more complex size models can be examined. The most important hardware size driver is the number of main components. We therefore propose to use the following simple formula to calculate the size of the hardware part  $s^{hw}$ .

$$s^{hw} = \sum_{i=1} \theta_1^{hw} \cdot c_i \quad (1)$$

### 2.3 Software Effort Points

For the hardware side of the Infrastructure Effort Point equation we should be able to apply the method to all possible situations, for the software side of the Infrastructure Effort Point equation we differentiate between two usage scenarios. In the first scenario the software is configured for the first or second time in the organisation (configuration engineering), whereas in the second scenario the organisation has a degree of experience in configuring the software (configuration roll-out). In a large deployment of software within an organisation one usually starts with configuration engineering and when all the required information about configuring the software has been gathered the project proceeds with configuration roll-out.

The difference between a first-time configuration and a repeat configuration is the familiarity with the software product. During a first installation one needs

to examine all configuration parameters to determine which parameters require adjustment. When one is familiar with a system one knows which parameters require adjustment and which factory settings are already correct.

For the software side of the Infrastructure Effort Point equation we distinguish one or two major size drivers. With configuration roll-out projects the major size driver is the number of configuration parameters. In recognising the effort required to determine which parameters to change, configuration-engineering has a second major size driver the total amount parameters in a group of parameter settings, that measures all available configuration parameters.

The major size driver for software configuration tasks is the number of configuration parameters  $p_i$  that require modification. During the installation of a software component the software is loaded from the installation medium to the target execution platform and simultaneously the settings of the components are set. The type of settings that determine the behaviour of a component can vary, broadly from installation directories, subsystem selection, user account creation, user settings to script files. Associated minor size drivers are: the type of values a parameter can store  $p_{i,j}^v$  and the input method that is required to change the parameter value  $p_{i,j}^t$ . The type of parameter values has an influence on the effect size of the task, because binary strings are much harder to enter and test compared to boolean parameters. The input method also has influence on the configuration size, as for example configuration using a gui is easier than configuring a system using a configuration file.

The other major size driver for configuration engineering tasks is the number of configuration parameters in the configuration group  $g_{i,j}$  that belong to the configuration parameter  $p_i$ . As configuring a system with only a few parameters is easier as configuring a system with a large number of parameters, the number of available configuration parameters also needs to be taken into account into the size equation. Therefore, all existing configuration parameters, that are seen by the IT team during installation and configuration are counted. The associated minor size drivers are the same as those described for the size driver number of configuration parameters.

To calculate the size for roll-out configuration tasks, the following model formula can be used:

$$s^{sw} = \sum_{i=1} \theta_1^{sw} \cdot p_i \quad (2)$$

To calculate the size for configuration-engineering tasks, the following model formula can be used:

$$s^{sw} = \sum_{i=1} (\theta_1^{sw} \cdot p_i) + \sum_{j=a} (\theta_2^{sw} \cdot g_{i,j}) \quad (3)$$

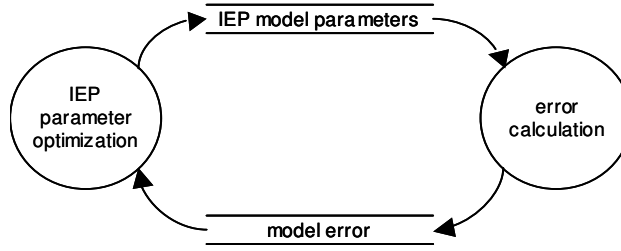


Fig. 2. Optimization of the IEP model parameters.

## 2.4 Infrastructure Effort Points

Having explained both the hardware and software part of the Infrastructure Effort Point measurement, we are ready to combine these two measurements into the overall Infrastructure Effort Point measurement.

To obtain the (total) Infrastructure Effort Point measurement the Hardware Effort Points and the Software Effort Points are added. To prevent one of the two measurements overruling the other measurement, a scaling factor  $\theta^{scale}$  is used. This leads to the following equation:

$$s^{hw} = s^{hw} + \theta^{scale} \cdot s^{sw} \quad (4)$$

## 3 Calibration Process

The purpose of the calibration is to find the IEP model parameters for an IEP size model. Together with the size model, the IEP model parameters make up the estimation equation of the Infrastructure Effort Points.

A good size metric should correlate well with development effort. We therefore define an optimal set of IEP model parameters to be those parameters that allow a cost estimation model to make the best possible prediction/explanation of the costs of a project. The advantage of using a cost estimation model as an intermediate to correlate size with development effort are twofold. First the cost estimation model can take phenomena as diseconomy of scale and schedule compression into account. The second advantage is that once the model has been calibrated using empirical data, other people can more easily recalibrate the model to their specific environment; the only need to recalibrate the estimation parameters.

To find the optimal calibration of the measurement model, empirical data about the infrastructural size drivers and the development effort is needed. The optimization process consists of the cyclic process of improving/modifying the initial IEP model parameters and consequently calculating error of the estimation algorithm (compared with the historical performance), see Fig. 2.

**Table 3.** Aggregated data from feasibility study.

<i>project name</i>	<i>effort</i>	<i>components</i>	<i>configuration roll-out</i>	<i>configuration engineering</i>	
			<i>parameters changed</i>	<i>parameters changed</i>	<i>parameters total</i>
Easy Icon Maker	7 min			0	1
Internet Explorer 5.5	10 min			13	20
MS Office 200 Professional	35 min			11	169
MS Virtual PC 2004	20 min			12	29
MS Windows ME	113 min			29	375
Printer installation	30 min	1			
Staff computer installation	120 min	1			
TCP/IP setup Windows ME	12 min			6	54
User info in MS Word 2000	9 min		5		

## 4 Feasibility Study

Having described the definition of the Infrastructure Effort Points and its calibration process, we describe the results of a preliminary feasibility study conducted to test the practicality of the data collection and its associated calibration process.

To test the data collection and calibration process, we collected measurements and effort data of nine projects that were collected in a controlled environment. The projects consisted of both hardware and software infrastructure projects. The aggregated results can be seen in Table 3<sup>2</sup>

The conclusion of the feasibility study is that it is possible to collect the required data in an efficient manner that does not disrupt the IT infrastructure project itself. All that is required is some degree of discipline in recording the steps during the project. Based on the data we are able to obtain a preliminary calibration of the data, however as this calibration is based only on nine projects it is not statistically significant.

## 5 Measurement Validation

Software metrics need to be validated to ensure that they measure what they purport to measure [20]. The validation of the metric should check whether the metric is valid and/or correct.

The validity of a software metric refers to its ability to provide measurements that can be used by practitioners in the context in which the metric was gathered. The correctness of a software metric refers to generic “laws” that govern measurements in general. An example of a correctness requirement is that addition of two measurement values should lead to a meaningful total. This requirement and other many correctness requirements are codified in measurement theory (see e.g. .

<sup>2</sup> The full dataset of the preliminary data collection is available in Microsoft Access-form, from the following address: <http://www.cs.vu.nl/reflection/infra-metrics/>.

As correctness requirements on software metrics have been extensively discussed in the academic literature (e.g. [21, chap.3]) they will not be discussed in more detail in this paper. In following section we pay more attention to the validity of the Infrastructure Effort Point metric. The validity requirements that are discussed in this section are: the focus aspect, the objectivity aspect, the timeliness aspect, the granularity aspect, and the genericity aspect.

## 5.1 Validity Requirements

The *focus aspect* of a software size metric determines whether the emphasis of the size of an IT solution lies on the size of the problem to be solved with an IT solution or on the size of the IT to create the solution. Certain complex function requirements on a system might take little implementation effort given the right tools whereas certain apparently simple functional problems might require a large amount of implementation effort because tool support is lacking. Metrics with a *value focus* pay more attention to the size of the problem and less on the size of the IT required to solve the problem. Metrics with a *work focus* pay more attention to the size of the IT solution as compared to the problem.

The IT infrastructure size metric has a work focus, as typical infrastructural IT projects are not free to choose which infrastructure to implement to support the functional requirements of the user. Lacking influence on the choice and availability of suitable IT infrastructure, it would not be fair to hold projects accountable for their implementation efficiency.

The *objectivity aspect* of a software size metric dictates whether the determination of the size of a software can be based partly on human judgment or can only be based on rules that can be interpreted in only a single manner. *Objective metrics* require only the application of clear rules and require no human judgment. *Subjective metrics* on the other hand do require human judgment and interpretation before a size can be determined. C.f. lines of codes [6] which can be counted automatically by a computer (a fully objective metric) with function points [5] (a partially subjective metric) which require a human function point counter to interpret the requirements specification. Size metrics should be as objective as possible, as subjectivity leaves room for disagreements on the real functional size and causes inaccuracies (e.g. [22]). The objectivity aspect of a metric is not a crisp distinction between objective and subjective, but a wide spectrum of nuances is possible.

The IT infrastructure size metric is an objective measure of the size of an IT infrastructural task. The determination of which hardware components are main components and which are sub-components does involve some subjective judgement. The same holds for the determination of the number of parameters in a parameter group, as the boundaries of the parameter group are not always very clear.

The *timeliness aspect* of a software size metric determines in which phase of the project one needs to be able to determine the size. Size measurements require information about a project or piece of software that becomes available in the course of the project. Certain information is available already at an early stage

of a project whereas other information only becomes available near the end of the project.

The IT infrastructure size metric is meant to be used for assessment and evaluation of methods and practices. This usually takes place at the end of a project, in contrast to practices such as estimation and project tracking that take place at an earlier phase of the project. As assessment and evaluation take place at the end of a project, it is not problematic if some of the required data only becomes available near the end of a project.

The *granularity aspect* of a software size metric refers to the aggregation level to which the metric is applied. The efficiency can be determined of a single project, of all work related to a single product type or of all work in an organizational unit. E.g. the efficiency of single network installation project can be measured (*project-level granularity*) or the efficiency of all networking operations in an organization based on cost per network point can be measured (*organizational-level granularity*). Between project-level and organizational-level metrics lie the *product-level* metrics that measure the efficiency of implementing a single type of product (e.g. a Linux server).

The IT infrastructure size metric needs to have a project-level granularity to explain in which contexts methods do work and in which contexts they do not work. Metrics with an organizational-level granularity would obscure the reasons why certain processes do or do not work.

The *genericity aspect* of a software size metric indicates how broad the applicability of the metric should be. *General-purpose* metrics can be applied to a broad range of software products, whereas *special-purpose* metrics can only be applied to a limited range of software products.

The IT infrastructure size metric needs to be a general-purpose metric. IT infrastructure includes a broad area of products. One can only meaningfully compare efficiency rates that are based on the same size measurements. It would therefore be beneficial if most IT infrastructure can be measured with the metric, allowing comparisons of the applicability of processes in different infrastructural domains.

## 6 Conclusions

In this paper we discuss the design and principles of the Infrastructure Effort Point metric for IT infrastructure projects. The metric is an objective metric that can be used to measure the size of IT infrastructure projects. It outperforms other existing size metrics for IT infrastructure in genericity and objectivity.

Infrastructure Effort Points can help to assess the effectiveness of processes and techniques, making it a valuable tool for process improvement for organizations that deliver IT infrastructure.

The Infrastructure Effort Point metric is unsuitable for the upfront estimation of a project's schedule or costs. The required information to feed into the size equation is available only at a late stage of the project. However with the aid of

additional techniques (e.g. PROBE [23, pp. 109–134]) it might well be possible to fruitfully use the metric for estimation as well.

The results look very promising, but some work still needs to be done. First, data about IT infrastructural project containing at least the IEP size drivers and the effort consumed by the project needs to be collected. This collected database will be used to calibrate the parameters in the Infrastructure Effort Point model and to analyse how good Infrastructure Effort Points correlate with the real expended effort.

## References

1. Abts, C., Boehm, B.W., Clark, E.B.: Observations on COTS software integration effort based on the COCOTS calibration database. In: Proceedings of the Twenty-Fifth Annual Software Engineering Workshop (SEW 25), NASA/Goddard Space Flight Center (2000) Available from: [http://sel.gsfc.nasa.gov/website/sew/2000/topics/CAbts\\_SEW25\\_Paper.PDF](http://sel.gsfc.nasa.gov/website/sew/2000/topics/CAbts_SEW25_Paper.PDF).
2. Morisio, M., Seaman, C.B., Basili, V.R., Parra, A.T., Kraft, S.E., Condon, S.E.: COTS-based software development: Processes and open issues. *Journal of Systems and Software* **61** (2002) 189–199
3. Carney, D., Long, F.: What do you mean by COTS? finally, a useful answer. *IEEE Software* **20** (2000) 83–86
4. Albrecht, A.J.: Measuring application development productivity. In: Proceedings of the Joint SHARE/GUIDE/IBM Applications Development Symposium. (1979) 83–92
5. Albrecht, A.J., Gaffney, Jr., J.E.: Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering* **9** (1983) 639–648
6. Park, R.E.: Software size measurement: A framework for counting source statements. Technical Report CMU/SEI-92-TR-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA (1992) Available from: <http://www.sei.cmu.edu/>.
7. Banker, R.D., Kauffman, R.J., Wright, C., Zweig, D.: Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment. *IEEE Transactions on Software Engineering* **20** (1994) 169–187
8. DeMarco, T.: *Controlling Software Projects: Management, Measurement & Estimation*. Yourdon Computing Series. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA (1982)
9. Basili, V.R., Boehm, B.: COTS-based systems top 10 list. *Computer* **34** (2001) 91–93
10. Morisio, M., Seaman, C.B., Parra, A.T., Basili, V.R., Condon, S.E., Kraft, S.E.: Investigating and improving a COTS-based software development process. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, I, IEEE Computer Society Press (2000) 32–41
11. Morisio, M., Torchiano, M.: Definition and classification of COTS: a proposal. In Dean, J., Gravel, A., eds.: Proceedings of the 1st International Conference on COTS Based Software Systems (ICCBBS 2002). Volume 2255 of Lecture Notes in Computer Science., Berlin, D, Springer-Verlag (2002) 165–175
12. Stensrud, E., Myrtveit, I.: Identifying high performance ERP projects. *IEEE Transactions on Software Engineering* **29** (2003) 398–416

13. Ardagna, D., Francalanci, C., Trubian, M.: A cost-oriented approach for infrastructural design. In: Proceedings of the 2004 ACM symposium on Applied computing, ACM Press (2004) 1431–1437
14. van Gorkom, V.: Service Point Analysis: A study on its effectiveness and usefulness. M.sc. thesis, CIBIT, Utrecht, NL (2002)
15. Raghavan, S., Achanta, V.: SYSPPOINT: Unit of measure for IT infrastructure project sizing. *Journal of Computing and Information Technology* **12** (2004) 31–46
16. Abts, C.M., Boehm, B.W.: COTS software integration cost modeling study. Technical report, Center for Software Engineering, University of Southern California, Los Angeles, CA, USA (1997)
17. Boehm, B.W., Abts, C.M., Bailey, B.: COCOTS software integration cost model: Insights and status (1999) Presented at: Ground System Architectures Workshop (GSAW-99).
18. Robinson, H., Sharp, H.: XP culture: Why the twelve practices both are and are not the most significant thing. In: Proceedings of Agile Development Conference, Washington, DC, USA, IEEE Computer Society Press (2003) 12–21
19. Verner, J., Tate, G.: A software size model. *IEEE Transactions on Software Engineering* **18** (1992) 265–278
20. Schneidewind, N.F.: Methodology for validation software metrics. *IEEE Transactions on Software Engineering* **18** (1992) 410–422
21. Fenton, N.E., Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach*. 2nd edn. International Thomson Computer Press, London, UK (1998)
22. Kemerer, C.F.: Reliability of function points measurement: a field experiment. *Communications of the ACM* **36** (1993) 85–97
23. Humphrey, W.S.: *A Discipline for Software Engineering*. Addison-Wesley Professional, Boston, MA, USA (1994)