Metrics and Measurements in Global Software Development

Maarit Tihinen and Päivi Parviainen
Digital Service Research
VTT Technical Research Centre of Finland
maarit.tihinen@vtt.fi
paivi.parviainen@vtt.fi

Rob Kommeren
Digital Systems & Technology
Philips,
The Netherlands
r.c.kommeren@philips.com

Jim Rotherham
Project Management Office
Symbio,
Finland
jim.rotherham@symbio.com

***Abstract*—Today products are increasingly developed globally in collaboration between subcontractors, third-party suppliers and in-house developers. However, management of a distributed product development project is proven to be more challenging and complicated than traditional single-site development. From the viewpoint of project management, the measurements and metrics are important elements for successful product development. This paper is focused on describing a set of essential metrics that are successfully used in Global Software Development (GSD). In addition, visualised examples are given demonstrating various industrial experiences of use. Even if most of the essential metrics are similar as in single-site development, their collection and interpretation need to take into account the GSD aspects. One of the most important reasons for choosing proposed metrics was their provision of early warning signs - to proactively react to potential issues in the project. This is especially important in distributed projects, where tracking the project status is needed and more complex. In this paper, the first ideas of GSD specific metrics are presented based on the common challenges in GSD practice.**

Keywords-metrics; measurements; global software development; distributed product development

I. INTRODUCTION

Global Software Development (GSD) is increasingly common practice in industry due to the expected benefits, such as lower costs and utilising resources globally. GSD brings several additional challenges to the development, which also affects the measurement practices, results and metrics interpretation. A current literature study showed that there is little research on GSD metrics or experiences of their use. This paper is enhanced and extended version of the ICSEA 2011 conference paper "Metrics in distributed product development" [1] where the metrics set had been successfully used in GSD were introduced. In this paper, the published metric set (with an example set of visualised metrics) was given with industrial experiences of their use. In addition, challenges faced during GSD are discussed from the viewpoint of metrics and measurements as well as potential GSD specific metrics.

Software metric is a valuable factor for the management and control of many software related activities, for example; cost, effort and schedule estimation, productivity, reliability and quality measures. Traditionally software measurement has been understood as an information gathering process. For example, software measurement is defined by [2] as follows: "The software measurements is the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products". The measurement data item consists of numeric data (e.g., efforts, schedules) or a pre-classified set of categories (e.g., severity of defects: minor, medium, major). Software metrics can consist of several measurement data items singly or in combination. Metric visualisation is a visual representation of the collected and processed information about software systems. Typically software metrics are visualised for presenting this information in a meaningful way that can be understood quickly. For example, visualising metrics through charts or graphs is usually easier to understand than long textual or numerical descriptions.

The main purpose of measurements and metrics in software production is to create the means for monitoring and controlling which provide support for decision-making and project management [3]. Traditionally, the software metrics are divided into process, product and resource metrics [4]. In the comprehensive measurement program, all these dimensions should be taken into consideration while interpreting measurement results; otherwise the interpretation may lead to wrong decisions or incorrect actions. A successful measurement program can prove to be an effective tool for keeping on top of the development effort, especially for large distributed projects [5]. However, many problems and challenges have been identified that reduce and may even eliminate all interests to the measurements. For example, not enough time is allocated for the measurement activities during a project, or not enough visible benefits are gained by the project doing the measurement work (e.g., data is useful only at the end of

project, not during the project). In addition, the "metric enthusiasts" may define too many metrics making it too time consuming to collect and analyse the data. Thus, it's beneficial [5] to define core metrics to collect across all projects to provide benchmarking data for projects, and to focus on measurements that come naturally out of existing practices and tools.

GDS development enables product development to take place independently of the geographical location, individuals or organizations. In fact, today the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers [6]. In practice distributed projects struggle with the same problems as single-site projects including problems related to managing quality, schedule and cost. Distribution only makes it even harder to handle and control these problems [7][8][9][10][11]. These challenges are caused by various issues, for example, less communication – especially informal communication – caused by distance between partners, and differences in background knowledge of the partners. That's why, in distributed projects the systematic monitoring and reporting of the project work is especially important, and measurement and metrics are an important means to do that effectively.

Management of a distributed product development project is more challenging than traditional development [12]. Based on an industrial survey [13], one of the most important topics in the project management in distributed software development is detailed project planning and control during the project. In GSD, this includes; dividing work by sites into sub-projects, clearly defined responsibilities, dependencies and timetables, along with regular meetings and status monitoring.

In this paper, a set of essential metrics used in GSD is discussed with experiences of their use. The main purpose is to introduce the selected metric set from the viewpoint of their proactive role in decision-making during globally distributed software development. The chosen metrics indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but requires special effort, distributed over sites and companies.

The amount of the metrics is intentionally kept as limited as possible. Also, the metrics should be such, that they provide online information during the projects, in order to enable fast reaction to potential problems during the project. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. Royal Philips Electronics is a global company providing healthcare, consumer lifestyle and lighting products and services. Digital Systems & Technology is a unit within Philips Research that develops first-of-a-kind products in the area of healthcare, well-being and lifestyle. The projects follow a defined process and are usually distributed over sites and/or use subcontractors as part of product development. Symbio Services Oy provides tailored services to organizations seeking to build tomorrow's technologies. Well-versed in a variety of software development methodologies and testing best practices, Symbio's specialized approaches and proprietary processes begin with product design and continue through globalization, maintenance and support. Symbio has built a team of worldwide specialists that focus on critical areas of the product development lifecycle. Currently, Symbio employs around 1400 people and their project execution is distributed between sites in the US, Sweden, Finland and China.

The metrics and discussion in the paper is based on GSD improvement work carried out during several years, in several research projects, including experiences from 54 industrial cases (see Parviainen [14], SameRoomSpirit Wiki [15]). This paper focuses especially on the experiences of two companies, Philips and Symbio.

The paper is structured as follows. Firstly, an overview of related work – available literature and its limitations related to measurements and metrics in distributed product development. This is introduced in Section II. In Section III, basic GSD circumstances with challenges are presented in order to explain the special requirements for measurements in GSD where the proposed metrics set is to be collected and utilised. In Section IV, measurement and metrics background and used terminology are introduced. In Section V, proposed metrics are presented using Rational Unified Process (RUP) [16] approach as a framework. The proposed metric set is presented with visualised examples and industrial experiences of their use. Furthermore, some GSD specific metrics are introduced in Section VI. Finally, discussion about metrics and their experiences is presented in Section VII and the conclusions are discussed in Section VIII.

II. RELATED WORK

There are several papers that discuss globally distributed software engineering and its challenges, for example, [5], [17] and [18]. Also, metrics in general and for specific aspects have been discussed in numerous papers and books for decades. However, little GSD literature has focused on metrics and measurements or even discusses the topic. Da Silva et al. [12] report similar conclusion based on analysis of distributed software development (DSD) literature published during 1999 – 2009: they state as one of their key findings that the "vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of DSD project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature." Bourgault et al. [19] reported similar findings, "Clearly, research into distributed projects' performance metrics and measurement needs more attention from researchers and practitioners so that it can contribute to the development and diffusion of well-designed management information systems."

The papers that have discussed some metrics for GSD usually focus on some specific aspect, for example, Korhonen and Salo [18], discuss quality metrics to support

defect management process in a multi-site organization. Misra [20] presents a cognitive weight complexity metric (CWCM) for unit testing in a global software development environment. Lotlikar et al. [21] propose a framework for global project management and governance including some metrics with the main goal to support work allocation to various sites. Lane and Agerfalk [22] use another framework as an analytic device to investigate various projects performed by distributed teams in order to explore further the mechanisms used in industry both to overcome obstacles posed by distance and process challenges and also to exploit potential benefits enabled by GDS. Similarly, Piri and Niinimäki [23] applied Word Design Questionnaire (WDQ) that consists of total of 21 sum variables in four categories (task characteristics, knowledge characteristics, social characteristics, and work context) to compare differences between the co-located and the distributed projects by metrics - "work design", "team dynamics", "teamwork quality", "project performance" and "individual satisfaction". These kinds of frameworks could be used to evaluate effectiveness of distributed team configuration during GSD projects as well. Peixoto et al. [17] discuss effort estimation in GSD, and one of their conclusions is that "GSD projects are using all kinds of estimation techniques and none of them is being consider as proper to be used in all cases that it has been used", meaning, that there is no established technique for GSD projects. In addition, some effort has also been invested in defining how to measure success of GSD projects [24], and these metrics mainly focus on cost related metrics and are done after project completion. These papers usually use common metrics that are not specific for GSD projects. For example, Ramasubbu and Balan [25] use 11 metrics (productivity, quality, dispersion, prevention QMA (Quality Management Approach), appraisal QMA, failure QMA, code size, team size, design rework, upfront investment and reuse), development productivity and conformance quality to evaluate how work dispersion effects to identified metrics. However, these metrics have not been used to gather information, indicators or experiences from ongoing distributed development.

Furthermore, only few papers discuss measurement tooling for GSD projects. Simmons [26] describes a PAMPA tool, where an intelligent agent tracks cost driver dominators to determine if a project may fail and tells managers how to modify project plans to reduce probability of project failure. Additionally, Simmons and Ma [27] discuss a software engineering expert system (SEES) tool where the software professional can gather metrics from CASE tool databases to reconstruct all activities in a software project from project initiation to project termination. Da Silva et al [28] discuss software cockpits from GSD viewpoint. They propose to examine various visualizations in the context of software cockpits, at-a-glance computer controlled displays of development-related data collected from multiple sources. They present three visualizations: (1) shows high-level information about teams and dependencies among them in an interactive world map, (2) displays the system design through a self-updating view of the current state of the software implementation, and (3) is a 3D visualization that presents an overview of current and past activities in individual workspaces.

The focus of this paper is to introduce a metrics set that creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. Furthermore, the paper gives several visualised examples of metrics that can be utilised while monitoring on-going GSD projects. The introduced metrics set can be seen as 'balanced score card', on which management can balance insights (~status) from time, effort, cost, functionality (requirements) and quality (tests) perspective.

III. BASIC GSD CIRCUMSTANCES WITH CHALLENGES

Parviainen [14] describes problems and challenges that are directly caused by the basic GSD circumstances. These challenges influence measurements and metrics and their interpretation during distributed software development. These challenges are mainly an intrinsic and natural part of GSD and they can either complicate globally distributed product development or even cause further challenges. The basic circumstances are:

- Multiple parties, meaning two or more different teams and sites (locations) of a company or different companies.
- Time difference and distance that are caused by the geographical distribution of the parties.

Problems caused by these circumstances include; issues such as unclear roles and responsibilities for the different stakeholders in different parties or locations, knowing the contact persons (e.g., responsibilities, authorities and knowledge) from different locations and establishing and ensuring a common understanding across distance. The basic GSD circumstances can also lead to poor transparency and control of remote activities as well as difficulties in managing dependencies over distance, problems in coordination and control of the distributed work and integration problems, for example. Problems may also be caused by basic circumstances in terms of accessing remote databases and tools or accordingly they may generate data transfer problems caused by the various data formats between the tools or different versions of the tools used by the different teams. The basic circumstances may also cause problems with data security and access to databases or another organisation's resources.

A commonly referenced classification for challenges caused by GSD is [29][30]:

- Communication breakdown (loss of communication richness)
- Coordination breakdown
- Control breakdown (geographical dispersion)
- Cohesion barriers (loss of "teamness")
- Culture clash (cultural differences).

Communication breakdown (loss of communication richness). Human beings communicate best when they are communicating face-to-face. In GSD, face-to-face communication decreases due to distance, causing misunderstandings and lack of information over sites. For example, communication over distance can lead to

misinterpretation because people cannot communicate well due to language barriers.

Coordination breakdown. Software development is a complex process that requires on-going adjustments and coordination of shared tasks. In geographically distributed projects, the small adjustments usually made in face-to-face contact do not take place or it is not easy to make adjustments. This can cause problem solving to be delayed or the project to go down the wrong track until it becomes very expensive to fix. GSD also sets additional requirements for planning, for example, the need for coordination between teams and the procedures and contacts for how to work with partners needs to be defined [31][32][33]. Coordination breakdown can also cause a number of specific problems; for example, Battin et al. [34] reported a number of software integration problems, which were due to a large number of independent teams. Wahyudin et al. [35] state that GSD demands more from project management. In addition to the project managers, the project members such as testers, technical leaders, and developers also need to be kept informed and notified of certain information and events that are relevant to their roles' objectives in timely manner which provides the conditions for in-time decision making.

Control breakdown (geographical dispersion). GSD means that management by walking around the development team is not feasible and, instead, telephones, email and other communication means (e.g., chat servers) must be used. These types of communication tools could be consider as less effective - not always providing a clear and correct status of the development site. Also, dividing the tasks and work across development sites, and managing the dependencies between sites is difficult due to the restraints of the available resources, the level of expertise and the infrastructure [34][36][37]. According to Holmstrom et al. [38], creating the overlap in time between different sites is challenging despite the flexible working hours and communication technologies that enable asynchronous communication. Lack of overlap leads to a delay in responses with a feeling of "being behind", "missing out" and even losing track of the overall work process.

Cohesion barriers (loss of "teamness"). In working groups that are composed of dispersed individuals, the team is unlikely to form tight social bonds, which are a key to a project's success. Lack of informal communication, different processes and practices have a negative impact on teamness [31][32][34]. Furthermore, fear (e.g., of losing one's job to the other site) has direct negative impact on trust, team building co-operation and knowledge transfer, even where good relationships existed beforehand. According to Casey and Richardson [39] fear and lack of trust negatively impact the building of effective distributed teams, resulting in clear examples of not wanting to cooperate and share knowledge with remote colleagues. Al-Ani and Redmiles [40] discuss the role that the existing tools can play in developing trust and providing insights on how future tools can be designed to promote trust. They found that tools can promote trust by sharing information derived from each developer's activities and their interdependencies, leading to a greater likelihood that team members will rely on each other which leads to a more effective collaboration.

Culture clash (cultural differences). Each culture has different communication norms. In any cross-cultural communication the receiver is more likely to misinterpret messages or cues. Hence, miscommunication across cultures is usually present. Borchers [41] discusses observations of how cultural differences impacted the software engineering techniques used in the case projects. The cultural indexes, power distance (degree of inequality of managers vs. subordinates), uncertainty avoidance (tolerance for uncertainty about the future) and individualism (strength of the relationship between an individual and their societal group), discussed by Hofstede [42], were found to be relevant from the software engineering viewpoint. Holmstrom et al. [38] discuss the challenge of creating a mutual understanding between people from different backgrounds. They concluded that often general understanding in terms of English was good, but more subtle issues, such as political or religious values, caused misunderstandings and conflicts during projects.

IV. MEASUREMENT BACKGROUND

In this section measurement background, the used terminology and traditional measurement methods, with GSD related challenges are introduced.

A. Traditional Metrics and Project Characteristics

Software measurements and metrics have been discussed since 1960's. The metrics have been classified many different ways. For example, they can be divided into basic and additional metrics [43] where basic metrics are size, effort, schedule and defects, and the additional metrics are typically metrics that are calculated or annexed from basic metrics (productivity = software size per used effort). The metrics can also be divided into objective or subjective metrics [43]. The objective metrics are easily quantified and measured, examples including size and effort, while the subjective metrics include less quantifiable data such as quality attitudes (excellent, good, fair, poor). An example of the subjective metrics is customer satisfaction. Furthermore, software metrics can be classified according to the measurement target, product, processes and resources [4]. Example metrics of product entities are size, complexity, reusability and maintainability. Example metrics of process entities are effort, time, number of requirements changes, number of specification/coding faults found and cost. Furthermore, examples of resource entities are age, price, size, maturity, standardization certification, memory size or reliability. These classifications, various viewpoints and the amount of examples merely prove how difficult the selection of metrics really can be during the project.

In addition to different ways of metrics classification, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparison should be done within the same kind of development projects, or the differences should be taken into account. Traditional project

characteristics are, for example; size and duration of a project, type of a project (development, maintenance, operational lifetime, etc.), project position (contractor, subcontractor, internal development etc.), type of software (hardware-related software development, application software, etc.) or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model etc.). Furthermore, different phases of development projects have to be taken consideration while analysing gathered measurement data.

B. Traditional Software Measurement and GSD

One of the most commonly used measurement methods at the end of 1990 and the beginning of 2000 was the Goal /Question /Metric (GQM) method. The GQM paradigm [3] represented a systematic approach for tailoring and integrating the objectives of an organisation into measurement goals and their step-wise refinement into measurable values. The GQM method was commonly known and was often used for searching and identifying organisations' strengths and weaknesses relating to the identified improvement goals. Furthermore, several assessment methods, for example CMMI [44] and SPICE (Software Process Improvement and Capability Determination, further known as a standard ISO/IEC 15504 Information technology — Process assessment), were generally used for identifying possible improvements areas and gaining knowledge of the software process of an organisation. In fact, the most of traditional measurements methods were based on expressions of the famous Shewhart cycle, called also the Deming cycle: PDCA (Plan–Do–Check–Act) [45]. The PDCA circle is an iterative four-step management method that is used in business for the control and continuous improvement of processes and products. The traditional methods used in software measurements were generally based on clearly defined and largely stabile processes that could be adjusted and improved. In those cases, the improvement actions were mainly done afterwards, for example, in the next project.

In GSD environment, where project stakeholders, work practices and development tools can vary by projects and partners, traditional measurement methods and actions are not adequate if they are used for process improvement purposes. There is little sense, if measurements only prove after the project what has happened during the project, because then it is too late to correct the situation. Furthermore, the lessons learned may not be suitable in the next projects. Overly large measurement programs with time consuming assessments are not worth paying the effort in dynamic GSD context. The traditional methods should be utilised for specific and well-aimed purposes. For example, the GQM method can be utilised while identifying new GSD specific metrics.

In GSD, development processes are dynamic and thus results of measurements and their interpretation vary. In this paper, GSD metrics used in the companies were focused on 'early warning' signals for the project and management. In a changing environment it's also an important aspect that the measurement data is easy to collect and that the metrics can be quickly calculated at regular intervals. Ease of use and speed are also central factors from metrics interpretation viewpoint. This also emphasises the importance of metrics visualisation. Interestingly, GSD literature has rarely focused on metrics and measurements or given experimental examples of successfully used metrics during GDS development.

C. Balancing Measurements

A Balanced Scorecard (BSC) is widely used for monitoring performance of an organisation towards strategic goals. The original BSC approach covers a small number of performance metrics from four perspectives, called as Kaplan & Norton perspectives: Financial, Customer, Internal Processes, Learning & Growth [46]. The BSC framework added strategic non-financial performance measures to traditional financial metrics to give managers and executives a more 'balanced' view of organizational performance. However, many early BSCs failed, because clear information and knowledge about the selection of measures and targets were not available. For example, organisations had attempted to use Kaplan & Norton perspectives without thinking about whether they were suitable in their situation. After that many improvements and enhancements have been completed on BSC approach. Since 2000, it has been described as a "Third Generation" of Balanced Scorecard designs. The BSC has evolved to be a strategic management tool that involves a wide range of managers in the strategic management process, provides boundaries of control, but is not prescriptive or constrictive and more importantly, removes the separation between formulation and implementation of strategy [47]. The BSC suggests that organisation should be viewed from four perspectives (Learning & Growth perspective, Business process perspective, Customer perspective, and Financial perspective) and metrics should be developed, data collected and analysed in relation to these perspectives.

Even if BSC are generally intended to deal with strategic issues, in this paper, the balancing of various perspectives of BSC has been emphasised. In fact, it has been proved that Practical Software Measurement and the Balanced Scorecard are both compatible and complementary [48]. In GSD context, decisions or actions taken based on the analysis of metrics and measurements collected from different development parties or stakeholders need to take specific the GSD factors into account as well.

D. Measurement Challenges in GSD

Even in the daily software development work, the measurements are still seen as unfamiliar or an extra burden for projects. For example, project managers feel it is time consuming to collect metrics for the organization (business-goal-related metrics), yet they need to have metrics that are relevant to the project. Furthermore, in many cases, not enough time is budgeted for measurements, and this is why it is very difficult to obtain approval from stakeholders for this kind of work [5].

Globally distributed development generates new challenges and difficulties for the measurements. For

example, the gathering of the measurements data can be problematic because of different development tools which have different versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. In addition, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that their comparison is impossible. Thus, the interpretation of measurements data is more complicated in GSD than one-site projects. This is why it is recommended to select a moderate amount of metrics. In this paper, we will present a set of metrics as well as examples of their visualisation possibilities to support decision making in GSD. Also industrial experiences about the metrics will be discussed.

The common metrics (effort, size, schedule, etc.) are also applicable for GSD projects. However, special attention may be needed in training the metrics collection, to ensure a common understanding of them (e.g., used classifications). In addition, as measurements also tend to guide people's behaviour, it is important to ensure that all are aware of the purpose of the metrics (i.e., not to measure individual performance), specifically in projects distributed over different cultures. In GSD content the automation of measurements is highly recommended to avoid misunderstanding - even if it is not easy to implement. The focus is to generate real-time information shown in a format that is easy and quickly interpreted. This means that great attention should be paid to metrics visualisation.

V. GENERIC MEASUREMENTS AND METRICS IN GSD

In this section, the metric set used in the companies is introduced. In addition, several visualised examples of proposed metrics are given and discussed. The metric set and their visualisation examples have been produced during the ITEA PRISMA (2008-2011) project [49]. The main goal of the PRISMA project was to boost productivity of collaborative systems development. One of the project's results was the Prisma Workbench (PSW), a tool integration framework [50]. PSW provides several real-time views into data that has been collected from various data sources even from separate stakeholders' databases. The PSW enabled the visualisation of metrics in GSD and collection of the experiences of their use. The work was done in close co-operation with industrial partners and experimental views were generated based on their needs or challenges. The original metrics were the same that the industrial partners had successfully used in their globally distributed projects, published in [1]. During the PRISMA project, the development of the PSW tool enabled further development of the proposed metrics set and their visualisation in co-operation with the industrial partners. The industrial partners had identified metrics, and defined their collection and visualisation. They had also tried the metrics in few projects to collect experiences. These experiences were then shared among the industrial partners of the project. The researchers analysed the measurements and experiences to find commonalities from these measurement practices. Results of this analysis was discussed in workshops with the companies, and updated based on the comments. This paper presents the results of this work. In following sub-sections, the developed example views are shown and discussed. Industrial experiences, opinions and ideas for improvement are also presented. The industrial experiences were gathered during the industrial cases by interviewing companies' personnel who had developed the metrics and measurement programs.

A. Rational Unified Process (RUP) Approach

Each phase in the lifecycle of a development project affects the interpretation of the metrics. Thus, in this paper, proposed metrics and visualisation examples are introduced by using commonly known approach of software development called Rational Unified Process (RUP). Also the processes used in the companies were similar to the RUP phasing, so it was chosen as a presentation framework for this paper. RUP is a process that provides a disciplined approach to assigning tasks and responsibilities within a development organisation. Its goal is to ensure the production of high quality software that meets the needs of its end-users within a predictable schedule and budget [16][51].

The software lifecycle is divided into cycles, each cycle working on a new generation of the product. RUP divides one development cycle in four consecutive phases [51]: (1) inception phase, (2) elaboration phase, (3) construction phase and (4) transition phase. There can be one or more iterations within each phase during the software generation. The phases and iterations of RUP approach are illustrated in following Figure 1.

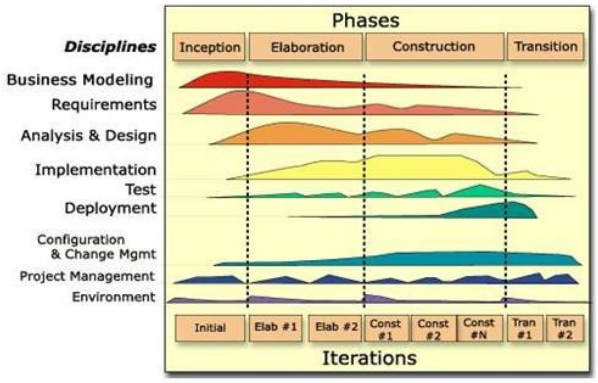

Figure 1. Phases and Iterations of RUP Approach [51]

From a technical perspective, the software development is seen as a succession of iterations, through which the software under development evolves incrementally [16]. From measurement perspective this means that some metrics can be focused on during one or two phases of the development cycle, and some can be continuous metrics that can be measured in all phases, and can be analysed in each iteration.

In this paper, the metrics are introduced according to the RUP phases. Each metric is presented in the phase where the

metric can be utilised in the first time or where the metric is seen to be the most relevant to measure, even if some metrics are relevant in several phases. In fact, many of the introduced metrics can be used also in the following product development phases. For each metric - a name, a notation and a detailed definition is introduced. The main goal is to offer a useful, yet reasonable amount of metrics, for supporting the on-time monitoring of the GSD projects. The indicators are supposed to be leading indicators rather than lagging indicators. For example, planned/actual schedule measurements should be implemented as milestone trend analysis which measures the slip in the first milestone and predicts the consequences for the other milestones and project end.

B. *Metrics and their Visualisation for Inception Phase*

During the inception phase, the project scope has to be defined and the business case has to be established. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones. Inception is the smallest phase in the project, and ideally it should be quite short. Example outcomes of the inception phase are a general vision document of the project's core requirements, main constraints, an initial use case model (10% -20% complete), and a project plan, showing phases and iterations [52]. Proposed metrics to be taken into consideration in this phase are introduced in Table I.

TABLE I. METRICS FOR THE INCEPTION PHASE

Metric	Notation	Definition
Planned Schedule	$D_{PLANNED}$	The planned Date of delivery (usually the completion of an iteration, a release or a phase)
Planned Personnel	# $FT_{PLANNED}$	The planned number of Full Time persons in the project at any given time
Planned Effort	$E_{PLANNED}$	The planned Effort for project tasks (/requirements) at any given time
Proposed Requirements	# Reqs	The number of proposed requirements.

The metrics Planned Schedule and Planned Personnel /Effort are mostly needed for comparison with actual schedule, personnel and effort, in order to identify lack of available resources as well as delays in schedule quickly. The amount of Proposed Requirements tells about the progress of the product definition.

Figure 2 shows how some of the proposed metrics can be utilised during product development for visualising the progress of project. The metric of progress status combines effort and schedule metrics in a visualised way. The first and top line (blue) in the Figure 2 is a cumulative planned effort over time calculated from project tasks. The next line, the red line describes the cumulative updated planned effort and accordingly, the green line describes the cumulative actual used effort over time summarised from project tasks. The bottom and last line in lilac shows the earned value that indicates the cumulative effort of completed tasks (/workproducts).

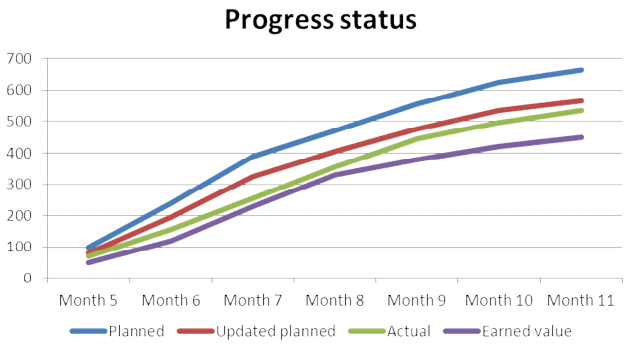

Figure 2. Visualised Metric: Progress Status

The graph visualises the project progress and easily gives several kinds of information as well as proactive insights, such as, is the project resourcing in place, and is the project completing work as planned. In the shown graph, it is a good signal that cumulative planned effort (blue line) is continuously above the cumulative updated planned effort (red line); it means the project is running on schedule. Another good signal is if actual used effort (green line) and earned value (lilac line) is relatively close to each others; it means that the results (~completed tasks) have been achieved with the used effort. The status in the Month 11 indicates that there are still several open tasks that are not completed even if actual used effort (green line) seems to draw closer to the cumulative updated planned effort (red line); this indicates a potential threat. Depending on project's phase (for example, in the middle phase or at the ending phase) corrective actions would be needed. The actions are not needed if the project is at the ending phase because the cumulative planned effort (blue line) is still clearly the upmost line.

Industrial comments

In the Philips company example, the Progress status metric has proven to give a timely insight in the actual consumption of effort compared to planned effort in large first-of-a-kind Consumer Electronics projects. The representation over time enables the ability to analyse trends, and take actions pro-actively. Moreover, the use of earned value gives insight in the effectiveness of the effort spent answering the question: "Does the effort spent contribute to realizing the agreed results?"

In the Symbio company example, indicators of earned value and tracking of unplanned work were seen as especially important from a management perspective. Unplanned work may yield a strong indication of a variety of causes early in the project, such as technical infeasibility or a lack of shared vision between project stakeholders. Accordingly, they identified that from a budget perspective, justifying workshops early in the project to shape a shared vision and collaborate on scoping project goals is often difficult to qualify for many stakeholders. It is a typical case that only when problems manifest, or a sharp trend in unplanned work is experienced will stakeholders react. Usually, remedying the problem requires unplanned trips to

put people together into the same room to hammer out solutions that essentially consume budget.

C. Metrics and their Visualisation for Elaboration Phase

During the elaboration phase a majority of the system requirements are expected to be captured. The purpose of the phase is to analyse the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. The final elaboration phase deliverable is also a plan (including cost and schedule estimates) for the construction phase. Example outcomes of the elaboration phase are; a use case model (at least 80% complete), a software architecture description, supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case, a revised risk list and a revised business case, and a development plan for the overall project. Proposed metrics to be taken into consideration in this phase are introduced in Table II.

TABLE II. METRICS FOR THE ELABORATION PHASE

Metric	Notation	Definition
Schedule: Planned /Actual Schedule	$\mathbf{D}_{PLANNED}$ $\mathbf{D}_{ACTUAL}$	The planned/actual Date of delivery (usually the completion of an iteration, a release or a phase)
Staff: Planned /Actual Personnel Planned /Actual Effort	$\mathbf{\#FT}_{PLANNED}$ $\mathbf{\#FT}_{ACTUAL}$ $E_{PLANNED}$ E_{ACTUAL}	The planned/actual number of Full Time persons in the project at any given time. The planned/actual Effort for project tasks (/requirements) at any given time.
Requirements -Drafted -Proposed -Approved -Not implemented	$\mathbf{\#Reqs}_{DRAFTED}$ $\mathbf{\#Reqs}_{PROPOSED}$ $\mathbf{\#Reqs}_{APPROVED.}$ $\mathbf{\#Reqs}_{NOT\_IMPL}$	The number (#) of - drafted requirements - proposed requirements - reqs approved by customer - not implemented reqs
Tests -Planned	$\mathbf{\#Tests}_{PLANNED}$	The number (#) of - planned tests
Documents: -Planned -Proposed -Accepted	$\mathbf{\#Docs}_{PLANNED}$ $\mathbf{\#Docs}_{PROPOSED}$ $\mathbf{\#Docs}_{ACCEPTED}$	The number (#) of planned /proposed /accepted documents to be reviewed during the project.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. The Staffing metric may explain deviations in the expected progress vs. the actual progress, both from a technical as well as from a schedule viewpoint. Note that those metrics that are more relevant to measure by iterations (effort and size) are introduced later (in Section E).

Figure 3 shows how some of the proposed metrics can be visualised in order to describe the project's status. The metric of requirements status combines the amount of planned effort with status of requirements' implementation over a time in the same graph. The bars summarise the amount of planned effort for the month. Each bar is composed from four different data relating to identified requirements as follows. The first block (green) describes a sum of planned efforts for all implemented requirements. The second block (grey) describes a sum of planned efforts for approved but not implemented requirements. The third block (blue) decribes a sum of planned efforts for proposed requirements and the last block (orange) shows a sum of planned efforts for drafted requirements.

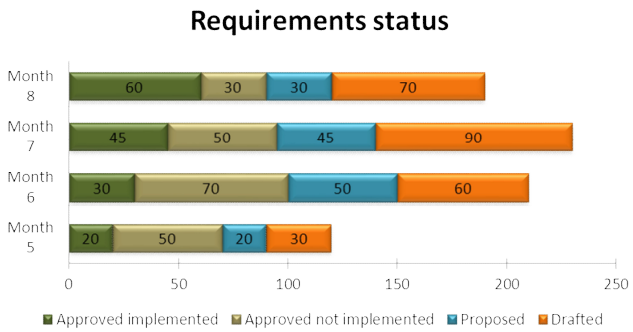

Figure 3. Visualised Metrics: Requirements Status

It is important to note, that the planned effort is used constantly, even for implemented requirements. This is due to keeping the baseline in order to enable comparing project situation over time, i.e., to be able to see the project trend with respect to planned work. The planned effort may be updated for the requirements during the project, if a new baseline is created. This information is then used together with the actuals, to see how well the planning has succeeded to help learning to estimate better.

The visualised metric "Requirements status" indicates several status information but also trend lines relating to requirements implementation, and is focused on showing the uncertainly of the project, for example how much more work maybe dedicated to be implemented in the project. In the example graph, a good signal is that the sum of planned efforts for implemented requirements seems to increase over time while the sum of planned efforts for approved, but not implemented requirements, seems to reduce. However, the sums of planned efforts for proposed and drafted requirements are still quite large in the Month 8, especially, while comparing them to the sums of planned efforts for approved requirements. This indicates that the project is in the beginning phase rather than in the ending phase. However, the interpretation needs other metrics information, such as "Progress status" or "Testing status" to make any decisions.

Industrial comments

In the Philips company example, the current projects lack insight into the satisfaction of requirements. This lack of insight concerns both the actual status of implementation of the requirements, as well as the expectation: "Up to what level the project will be able to satisfy its requirements, and if not, what are measures to accomplish that?" The (leading) indicator as proposed in this document seems to be a good answer to this problem. The metric has been introduced in a few (one-roof) projects yet and initial results seem promising. However, no data with experiences on a metric like this have been collected yet.

According to Symbio's practice, when looking to exit an elaboration phase, product owners should pay special

attention to the coverage of requirements affecting architecture to ensure the construction phases run more to plan as the team sizes may scale and involve more sites. Whilst iterative development can be seen as promoting elaboration of requirements later in the lifecycle, core functions that separate the project output from competition should be conceptualized and approved for implementation. Project managers may consider implementation of these differentiating use cases to be made geographically or temporally close to the project owner. Non-approved requirements should be managed accordingly and not planned for implementation off-site until they are suitably elaborated and accepted into the development roadmap. Misunderstanding of the requirements needs to be minimalized if the team size and development sites scale during construction phases otherwise projected cost savings from multi-site development can be quickly eliminated.

D. Metrics and their Visualisation for Construction Phase

Construction is the largest phase in the project. During the phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. Example outcomes of the phase consist of a software product integrated on the adequate platforms, user manuals, and a description of the current release. Proposed metrics to be taken into consideration in this phase are introduced in Table III.

TABLE III. METRICS FOR THE CONSTRUCTION PHASE

Metric	Notation	Definition
Planned /Actual Schedule Planned /Actual Personnel	$\mathbf{D}_{\text{PLANNED}}$ $\mathbf{D}_{\text{ACTUAL}}$ $\mathbf{\#FT}_{\text{PLANNED}}$ $\mathbf{\#FT}_{\text{ACTUAL}}$	Defined in the elaboration phase.
Requirements: -Proposed -Approved -Not implemented -Started -Completed	 $\mathbf{\#Reqs}_{\text{PROPOSED}}$ $\mathbf{\#Reqs}_{\text{APPROVED.}}$ $\mathbf{\#Reqs}_{\text{NOT\_IMPL}}$ $\mathbf{\#Reqs}_{\text{STARTED}}$ $\mathbf{\#Reqs}_{\text{COMPLETED}}$	The number (#) of - proposed requirements - reqs approved by customer - not implemented reqs - reqs started to implement - reqs completed
Change Requests: -New CR -Accepted -Implemented	 $\mathbf{\#CRs}_{\text{NEW}}$ $\mathbf{\#CRs}_{\text{ACCEPTED}}$ $\mathbf{\#CRs}_{\text{IMPL.}}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented
Tests: -Planned -Passed -Failed -Not tested	 $\mathbf{\#Tests}_{\text{PLANNED.}}$ $\mathbf{\#Tests}_{\text{PASSED}}$ $\mathbf{\#Tests}_{\text{FAILED}}$ $\mathbf{\#Tests}_{\text{NOT TESTED}}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
Defects -by Priority: e.g., Showstopper, Medium, Low	$\mathbf{\#Dfs}_{\text{PRIORITY}}$	The number (#) of - defects by Priority during the time period
Documents: -Planned -Proposed -Accepted	 $\mathbf{\#Docs}_{\text{PLANNED}}$ $\mathbf{\#Docs}_{\text{PROPOSED}}$ $\mathbf{\#Docs}_{\text{ACCEPTED}}$	Defined in the elaboration phase.

Note that those metrics that are continuously measured are introduced later (in Section E).The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Defect metrics describe both the progress of testing as well as the maturity of the product.

In the construction phase, all components and features are developed and integrated into the product. In addition, they are also thoroughly tested, so there are many simultaneous actions that can be implemented by multiple partners or/and in different locations in GSD. This is why the metrics interpretation needs to be done very carefully by utilising indicators from different data sources and from different partners. In this subsection two metrics: "Budget status" and "Testing status" are introduced with discussion about indicators and proactive signals that they provide.

The visualisation of Budget status combines cost, requirements and defects metrics in the same graph shown in Figure 4.

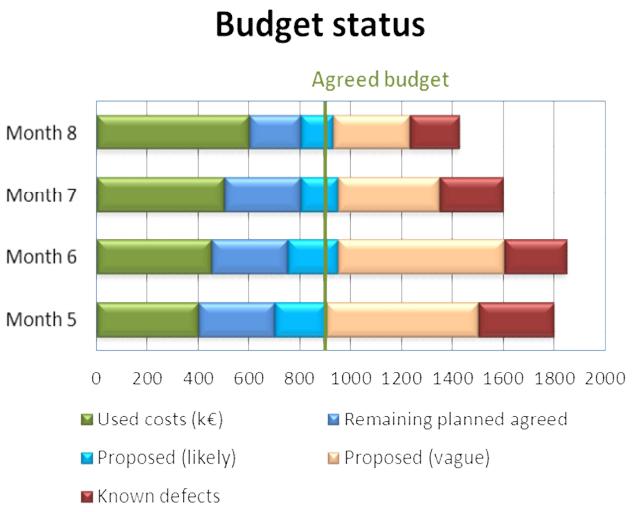

Figure 4. Visualised Metrics: Budget Status

The Budget status graph shows actual costs of the project in portion with the agreed budget over a time period. The metric also gives several indicators of estimated prospective costs in each month. The bars summarise amount of costs for the month, and each bar is composed from five different cost-related data. The first block (green) describes actual cumulative costs of the project. The agreed budget for the project is shown clearly as a green line in the middle of the graph. The second block (blue) describes remaining planned cost based on effort estimated for requirements that have been accepted for implementation but not yet implemented. The third block (light blue), in the middle of the bar, indicates proposed cost that can be seen very likely costs for the project. These costs are based on effort estimated for the proposed requirements that are estimated likely to be implemented, for example, a customer will want them. The fourth block (orange) describes proposed but vague costs for

the project. These costs are based on effort estimated for the proposed requirements which the likeliness for implementation is not known. Instead, the fifth block (red) indicates very potential costs for the project, so-called "Known defects" costs. The costs are based on effort estimated to be needed to fix the known critical, major or average defects. In the example graph, the Budget status metric in Figure 4, the project's costs will overrun the agreed budget.

Industrial comments

At Philips, the current applied budget metrics generally give a clear understanding in the actual budget consumption, but are poor in predicting budget consumption for the remainder of the project. The metric suggested allows for trend analysis and by that extrapolation to the future, resulting in better prediction of the budget consumption for the remainder of the project. This will improve the projects' and the management's insight into the project and enable them to take required measures in a timely fashion, as appropriate. The metric has not yet been applied in our projects.

In Symbio, managers will often track cost against budget throughout construction for project sponsors, but earned value becomes increasingly more important in the latter stages of the lifecycle. Earned value can be tracked with relative ease if defined requirements are quantified for business importance. Product backlogs imply the importance by a requirement's position in the backlog; however some backlogs may include other items than requirements such as operational tasks for deployment and so on. To compensate all project requirements (both functional and non-functional) can be attributed with a business value, its value based in comparison to the cumulative value of all project requirements. When a requirement is delivered its value is added to cumulative total to provide an earned value delivered by the project. This approach is ideal if the backlog of the product development stabilizes throughout construction. However significant changes in the business value of requirements will weaken the importance of tracking this metric over time. Also this metric requires the project team and stakeholders to agree upon a "definition of done" which can be very difficult, and even more so if the accepting and implementing parties are different entities or located in different sites.

The metric of Testing status combines effort, requirements and test metrics in a same graph. The Testing status metric visualises the progress of testing phase by collecting data from various phases. The bars in the graph summarise efforts relating to tests in each month. Each bar is composed from four different sums of efforts. The first block (green) describes a sum of efforts for tested requirements. The second block (blue) describes a sum of efforts for requirements for which test case is available, and accordingly, the third block (purple) describes a sum of efforts for requirements for which test cases are not available. The last, the fourth block (red) is a very proactive indicator describing a sum of effort estimated for uncertain requirements. Figure 5 shows the visualisation of Testing status metric.

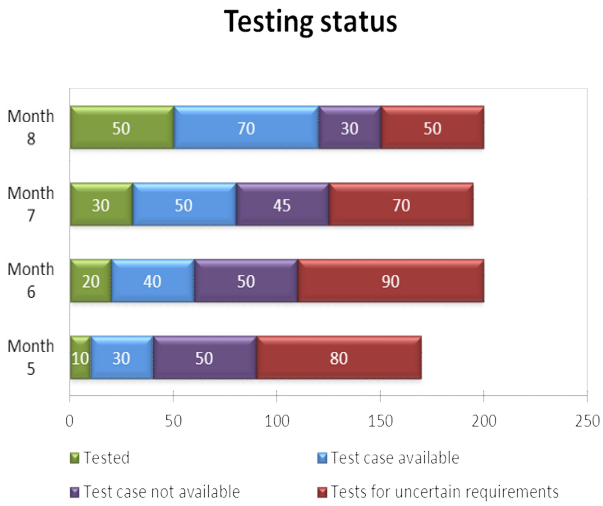

Figure 5. Visualised Metrics: Testing Status

Even if "Testing status" shows easily how 'mature' the testing phase is the metric requires other metrics – such as the before introduced metrics: Budget status, Progress status and Requirements status – make conclusions based on the data.

Industrial comments

According to Philips, one of the most important indicators of a development project is insight in what will be the status of the product at the delivery time - what will the product actually contain and what is the quality of those contents? This metric is an effective means to get early insight in the status of the product by the end of the project. Moreover, the test status trend analysis helps to initiate timely measures to work towards an agreed project result. The metric has been applied in a single project at Philips and results were promising - it really improved the insight of project, management and customer in the status of the product-under-construction and better understanding of what could be expected by the end of the project.

According to Symbio, earned value is especially invaluable in the close down phases of a project. Projects may deteriorate into loss making, unplanned iteration as stakeholders become overly conscious on metrics of requirements coverage. This situation is can be further exacerbated if the value of requirements is not continually reviewed and communicated to all stakeholders throughout the project.

E. Metrics for Transition Phase

The final project phase of the RUP approach is transition. The purpose of the phase is to transfer a software product to a user community. Feedback received from initial release(s) may result in further refinements to be incorporated over the course of several transition phase iterations. The phase also includes system conversions, installation, technical support, user training and maintenance. From measurements viewpoint the metrics identified in the phases relating to schedule, effort, tests, defects, change requests and costs are still relevant in the transition phase. In addition, customer

satisfaction is generally gathered in the transition phase, and post-mortem analysis carried out.

F. Metrics for Iterations

Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release. Each release is accompanied by supporting artifacts: release description, user's documentation, plans, etc. Although most iterations will include work in most of the process disciplines (requirements, design, implementation, testing) the relative effort and emphasis will change over the course of the project. Proposed metrics for each iteration to be taken into consideration, are introduced in Table IV.

TABLE IV. METRICS FOR ITERATIONS

Metric	Notation	Definition
Effort: -Planned Effort -Actual Effort	$\mathbf{E}_{PLANNED}$ $\mathbf{E}_{ACTUAL}$	The planned/actual effort required of any given iteration of the project.
Size: -Planned size -Actual size	$\mathbf{SIZE}_{\text{PLANNED}}$ $\mathbf{SIZE}_{\text{ACTUAL}}$	The planned /actual size of each iteration can be measured as SLOC (Source Lines of Code), Points or any other commonly accepted way.
Cost: -Budgeted -Expenditure	$\mathbf{COST}_{\text{BUDGET}}$ $\mathbf{COST}_{\text{ACTUAL}}$	The budgeted cost /actual expenditure for any given iteration.
Velosity: -planned /actual story points	$\mathbf{\#PTS}_{\text{PLAN}}$ $\mathbf{\#PTS}_{\text{ACT}}$	How many story points are planned to be /actually implemented of any given iteration of the project.
Productivity:	$\mathbf{E}_{ACTUAL}$ / $\mathbf{\#PTS}_{\text{ACT}}$	Use effort per acutally implemented story points for each sprint /iteration

All of these metrics provide indications of the project progress and reasons for deviations should be analysed. These metrics should be analysed together with other metrics results (presented in Tables I-III) in order to gain a comprehensive picture of the status.

VI. SPECIFIC MEASUREMENTS AND METRICS IN GSD

Section V discussed metrics, which are not specific for GSD, but they provide valuable information to follow a GSD project progress. So, in GSD, metrics can be similar or same as in single-site development. However, in order to prevent potential problems during distributed projects some specific GSD metrics could be added to be used together with the metrics presented in Section V. These metrics should be focused on the specific challenges in GSD that were presented in general level in Section III and they would help to quickly detect the GSD related source of problems that are identified in the metrics presented in Section V.

Measuring the generic GSD challenges (Section III) is difficult, and in fact, measuring the challenges does not provide clear value from project monitoring viewpoint. It is more beneficial to follow and detect the symptoms that indicate problems in the GSD practice. Example problems [14] caused by lack of communication, coordination breakdown, and different backgrounds include, for example, ineffective use of resources as competences are not known from other sites, obstacles in resolving seemingly small problems and faulty work products due to a lack of competence or background information. These causes can also lead to a lack of transparency in the other parties' work, misunderstood assignments and, thus, faulty deliveries from parties, delays caused by waiting for the other parties' input and duplicate work or uncovered areas. Further problems that can be caused by these issues include differences in tool use or practices in storing information, misplaced restrictions on the access to data and unsuitable infrastructure for the distributed setting.

Example problems [14] caused by lack of teamness and lack of trust include hiding problems and unwillingness to ask for clarification from others, expending a lot of effort in trying to find that the cause of problems (defects) has occurred in the other parties' workplace, an unwillingness to help others and an unwillingness to share information and work products until specifically requested to do so. These causes may also appear as difficulties in agreeing about the practices to be used and then not following the process and practices as agreed, for example. Further problems caused by these issues include the use of other tools than those agreed to for the project and plentiful technical issues that hinder communication and use of the tools, as agreed.

The following problems are among the most common ones in companies GSD practice (based on 54 industrial cases during several research projects):

1. unclear responsibilities and escalation channels,
2. unavailability of information timely for all who need it,
3. unclear information and misunderstandings (for example of requirements and task assignments),
4. problem hiding,
5. non-communicated and unexpected changes,
6. lack of visibility and transparency of all sites work and progress,
7. faulty and/or delayed (internal) deliveries, and
8. sub optimal use of resources.

Next we discuss potential measurements to indicate as early as possible if these problems are present. These proposals have not been applied in practice, yet. Instead, their implementations and possible selections were discussed with industrial partners.

Relating to problems 1-3, a measurement could be a short questionnaire asking the project members if they know their responsibilities and when and to whom to escalate problems, and is the required information available and clear. In addition, from GSD viewpoint, a potential measurement could be related to time spent idling (a team member is waiting because of wrong, incorrect or missing information or input from other members) or percentage of unplanned work (a team member is working with unplanned or duplicated tasks).

For problems 4 – 6, a measure is amount and type of communication over sites. For example, communications activeness could be monitored via metrics like amount of status reports, meeting memos, chats, calls between

locations, etc. Communication activeness is especially important between distributed teams where their development tasks are highly coupled or dependent on deliveries and results of each other. For example, silence or communication only via documents (official reports) can be an indicator of problem, whereas active informal communication over sites indicates active discussion of work at hand. In the worst case in GSD, lack of face-to-face communication can lead to "reportmania" where communication is handled only through large amount of documents. Long textual descriptions can be easily omitted or alternatively misunderstood because of high amount of effort and time required for adopting the content.

For problem 7, metrics related to defects and schedule are relevant and for problem 8, a potential measurement is time spent idling and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when a team is not performing. Also, communication related metrics are valuable for these problems.

The metrics relating to team trust, project commitment and team identifications describes team dynamics that can provide lot of explaining information for the problems in GSD project. Some indicators, such as how many people have left from the project, refer the individual satisfaction as well as project commitment. Because software development is fundamentally team oriented action [53], metrics relating to team dynamics and teamwork quality is highly recommended to monitor in GSD. Potential metrics are related to communication, tasks coordination, balance of member contributions, mutual support, effort and cohesion as introduced by Hoegl and Gemuenden [54]. Examples of questions are as follows:

- Communication: Is there sufficient frequent, informal, direct, and open communication?
- Coordination: Are individual efforts well-structured and synchronised within the team?
- Balance of member contributions: Are all team members able to bring in their expertise to its full potential?
- Mutual support: Do team members help and support each other in carrying out their tasks?
- Effort: Do team members exert all efforts to team tasks?
- Cohesion: Are team members motivated to maintain the team? Is there team spirit?

These questions can be used to measure team dynamics and team work quality during a GSD project.

VII. DISCUSSION

A. GSD Metrics

As discussed, little focus has been paid on GSD metrics in the literature. In fact, the research has been focused on clarifying differences between collocated and distributed projects and also, identifying variables that differ the most. Although this kind of approach is important for gaining knowledge about the issues that need to be monitored in GSD, a specific focus on the metrics and their collection and analysis is also needed. For example, project performance is even more complicated and multi-level concept to measure in GSD than in single-site. It concerns team members' individual performance, teamwork performance and tasks performance as well as management performance. Bourgault et al. [19] pointed out that distributed projects' performance metrics and measurement needs more attention so that well designed management information systems could be developed in order to create effective monitoring systems for distributed projects. This kind of development was seen as necessary to provide decision makers with dynamic, user-friendly information system that would support management activities, not only for project managers, but also for top managers. However, the issue of performance metrics in the context of distributed projects needs to be investigated in more detail. Furthermore, a dispersion of work has significant effects on productivity and, indirectly, on the quality of the software. However, it is currently difficult to specify metrics, measurement processes and activities that best suit different companies and specific GSD circumstances. We have presented a first step towards taking into account the specific aspects of GSD in measurement programs, but more work is needed. For example, specific GSD metrics are currently collected and processed manually, thus requiring extra and error prone effort. In the world of the hectic and dynamic GSD practice, the metrics collection and visualisation should also be automated to be valuable in large-scale use. The automation is an important issue for further research.

B. Industrial Viewpoint

The metrics presented in Section V were common for both of the companies. Although the metrics were chosen independently by both companies, the reasoning behind choosing these metrics was similar. An important reason was to come from a re-active into a pro-active mode, for example to introduce 'early warning' signals for the project and management. Specifically these metrics have been chosen as they indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort, distributed over sites and companies. Accordingly, the metrics set can be seen as a 'balanced score card', on which management can take the right measures, balancing insights from time, effort (e.g., staffing), cost, functionality (requirements) and quality (tests) perspective.

An important aspect was also that the metrics are easy to capture and that they can be captured from the used tools "for free", or can be quickly calculated at regular intervals. Costs and budgets are good examples of metrics that can be easily captured from the tools. This is also important from GSD viewpoint, as automated capturing reduces the chance of variations caused by differences in recording the metrics data in different sites. Neither of the companies use metrics based on lines-of-code as they did not find it to be a reliable indicator of progress, size or quality of design.

It can be seen that the metrics are quite similar as in single-site development. However, the metrics may be analysed separately for each site, and comparisons between sites can thus be made in order to identify potential problems early. On the other hand, it is important to recognise that some metrics correlate with each other, for example, metrics relating to tests correlate with metrics about requirements, and that needs to take consideration while analysing. In general, the interpretation of project's comprehensive status needs various metrics information – like Requirements status, Progress status, Testing status and Budget status – for making conclusions based on the data. In addition, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires 'super-balancing' - how to come to the right corrective action if for instance, on the one side, the % of not accepted requirements is high, and on the other side, the # of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, relating to subjective metrics, such as effort estimation, differences between backgrounds of the people (cultural or work experience) in different sites may affect the result.

The companies also use the measurement results to gain insight into why a measure varies between similar single-site and multi-site projects in order to try to reduce potential variances. This also partially explains the use of the same metrics as single-site development. This was experienced by the representative of Symbio: "*These points presented should be by now well known. From an economic perspective these points must be considered when evaluating and comparing costs of different project models of delivery.*", and "*Benchmarking and tracking of historical data across the entire project portfolio is still only an initial step to shape more informed cost estimations when composing project teams with distributed elements. Continuous effort is required not only in definition and capture of metrics but also in the effects on working practices in general.*"

Furthermore, the challenges in communication and dynamics of distributed teams mean that working practices need to be addressed continuously as impressed by Symbio representative: "*Often a practical solution to working procedures can result in compensation for potential lost productivity. For example a testing team in China lags their working week by one day (Tuesday to Saturday) in order to test the results from an implementation team in Finland (working Monday to Friday). In this example the Finland team agrees to ensure continuous integration in order to not block the testing team. If these two practices have a positive effect on productivity when compared against similar project models, future cost estimations should then be benchmarked on the new working practices.*" However, in addition to metrics results, paying close attention and acting on feedback is as important, if not more important than drawing strong conclusions from metrics alone.

Currently, both companies are in process of revamping their metric usage, but feel confident that the metrics introduced in this paper are the right ones. This was pointed out by Philips by the following: "*Applying the metrics suggested in this document to the parties involved in the GSD project already gives better insight in the relative performances of the groups, and enables to take measures over time (e.g., systematically improve a party's performance, or replace it). We have applied detailed effort consumption metrics to our single-roof and multi-side development projects. Those metrics learned that staff of multi-side projects spend significantly more time on things they call 'communication' or 'overhead' (up to 50%!). Our understanding of the matter is that no new metric needs to be 'invented' for that: standard effort distribution metrics would do. The main challenge is to have it introduced in a systematic way, with the same understanding and interpretation of the metrics by the parties involved. Especially the first element is often a challenge: third parties are often reluctant to provide this level of transparency of their performance.*"

Both companies are careful in introducing new metrics, as it is well known that too many metrics lead to overkill and rejection by the organization, and do not provide the right insights and indication for control measures. Easy implementation and by that, easy acceptance is the most crucial thing to get these metrics as established practice within the company. However, the few specific GSD metrics presented in Section VI are intended to be used together as the proposed metrics set. These additional metrics should be focused on measuring the project performance, especially task and team performance in GSD.

VIII. CONCLUSION

The management of the more and more common distributed product development project has proven to be more challenging and complicated than traditional one-site development. Metrics are seen as important activities for successful product development as they provide the means to effectively monitor the project progress. However, defining useful, yet reasonable amount of metrics is challenging, and there is little guidance available for a company to define metrics for its distributed projects.

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools and their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. Furthermore, interpretation and decision-making based on the measurement results require that the distributed development implications are taken carefully into consideration.

This paper focused on describing a set of metrics that is successfully used in industrial practice in GSD and given examples of their visualisation with industrial experiences of their use. These metrics, are aimed especially to provide the means to proactively react to potential issues in the project, and are meant to be used as a whole, not interpreted as single information of project status. The basic GSD circumstances with challenges are discussed from viewpoints of metrics

and measurements in order to create awareness and knowledge of potential GSD specific metrics.

The metrics presented in the paper were common for both of the companies. Based on experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed at regular intervals. Also, one of the most important reasons was that these metrics were aimed especially to provide the means to proactively react to potential issues in the project. The balancing insights from time, effort, cost, functionality and quality was also seen as very important aspect.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024 [49]. The authors would like to thank the support of ITEA [55] and Tekes (the Finnish Funding Agency for Technology and Innovation) [56].

REFERENCES

[1] M. Tihinen, P. Parviainen, R. Kommeren and J. Rotherham, "Metrics in distributed product development," In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11), Barcelona, Spain, 2011, pp. 275-280.

[2] R. Van Solingen and E. Berghout, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill, 1999.

[3] V. R. Basili, Software modeling and measurement: The Goal /Question/Metric paradigm. Computer Science Technical Report CS-TR-2956, UNIMACS-TR-92-96, University of Maryland at College Park, Sep. 1992, pp. 1-24.

[4] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co. Boston, MA, USA, 1998.

[5] M. Umarji and F. Shull, "Measuring developers: Aligning perspectives and other best practices," IEEE Software, vol. 26, (6), 2009, pp. 92-94.

[6] J. Hyysalo, P. Parviainen and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," In Proceedings of 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.

[7] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," In Proceedings of Future of Software Engineering FOSE '07, IEEE Computer Society, 2007, pp. 188-198.

[8] J. D. Herbsleb, A. Mockus, T. A. Finholt and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, 2000, pp. 319-328.

[9] M. Jiménez, M. Piattini and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," Advances in Software Engineering, vol. Jan-2009, (No. 3), 2009, pp. 1-16.

[10] S. Komi-Sirviö and M. Tihinen, "Lessons learned by participants of distributed software development," Knowledge and Process Management, vol. 12, (2), 2005, pp. 108-122.

[11] M. Tihinen, P. Parviainen, T. Suomalainen, K. Karhu and M. Mannevaara, "ABB experiences of boosting controlling and monitoring activities in collaborative production," In Proceedings of the 6th IEEE International Conference on Global Software Engineering (ICGSE'11) Helsinki, Finland, 2011, pp. 1-5.

[12] F. Q. B. da Silva, C. Costa, A. C. C. França and R. Prikladinicki, "Challenges and solutions in distributed software development project management: A systematic literature review," In Proceedings of International Conference on Global Software Engineering (ICGSE2010), IEEE, 2010, pp. 87-96.

[13] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," In Proceedings of the15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.

[14] P. Parviainen, "Global software engineering. challenges and solutions framework," Doctoral Dissertation, VTT Science 6, Finland, 2012, pp. 106 p. + app. 150 p.

[15] Prisma-wiki, SameRoomSpirit wiki homepage. URL: http://www.sameroomspirit.org/index.php/Main_Page (Accessed 19.12.2012).

[16] P. Kruchten, The Rational Unified Process: An Introduction. Addison-Wesley Professional, 2004.

[17] C. E. L. Peixoto, J. L. N. Audy and R. Prikladnicki, "Effort estimation in global software development projects: Preliminary results from a survey," In Proceedings of International Conference on Global Software Engineering, IEEE Computer Society, 2010, pp. 123-127.

[18] K. Korhonen and O. Salo, "Exploring quality metrics to support defect management process in a multi-site organization - A case study," In Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2008, pp. 213-218.

[19] M. Bourgault, E. Lefebvre, L. A. Lefebvre, R. Pellerin and E. Elia, "Discussion of metrics for distributed project management: Preliminary findings," In Proceedings of the 35th Annual Hawaii International Conference on System Sciences HICSS'02, IEEE, 2002, 10 p.

[20] S. Misra, "A metric for global software development environment," In Proceedings of the Indian National Science Academy 2009, pp. 145-158.

[21] R. M. Lotlikar, R. Polavarapu, S. Sharma and B. Srivastava, "Towards effective project management across multiple projects with distributed performing centers," In Proceedings of IEEE International Conference on Services Computing (CSC'08), IEEE, 2008, pp. 33-40.

[22] M. T. Lane and P. J. Ågerfalk, "Experiences in global software development - A framework-based analysis of distributed product development projects," In Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009). 2009, pp. 244-248.

[23] A. Piri and T. Niinimaki, "Does distribution make any difference? quantitative comparison of collocated and globally distributed projects," In Proceedings of the Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW'11), 2011, pp. 24-30.

[24] B. Sengupta, S. Chandra and V. Sinha, "A research agenda for distributed software development," In Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 731-740.

[25] N. Ramasubbu and R. K. Balan, "Globally distributed software development project performance: An empirical analysis," In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference aNd the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), ACM, 2007, pp. 125-134.

[26] D. B. Simmons, "Measuring and tracking distributed software development projects," In Proceedings the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003). IEEE, 2003, pp. 63-69.

[27] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," In Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), IEEE, 2006, pp. 33-38.

[28] I. A. da Silva, M. Alvim, R. Ripley, A. Sarma, C. M. L. Werner and A. van der Hoek, "Designing software cockpits for coordinating distributed software development," In the First Workshop on

Measurement-Based Cockpits for Distributed Software and Systems Engineering Projects, 2007, pp. 14-19.

[29] E. Carmel, Global Software Teams: Collaborating Across Borders and Time Zones. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[30] E. Carmel and P. Tija, Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce. Cambridge University Press, the United Kingdom, 2005.

[31] D. E. Damian and D. Zowghi, "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations," In Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), 2003, 10 p.

[32] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," IEEE Transactions on Software Engineering, vol. 29, (6), 2003, pp. 481-494.

[33] M. Paasivaara and C. Lassenius, "Collaboration practices in global inter-organizational software development projects," Software Process: Improvement and Practice, vol. 8, (4), 2003, pp. 183-199.

[34] R. Battin, R. Crocker, J. Kreidler and K. Subramanian, "Leveraging resources in global software development," IEEE Software, vol. 18, (2), 2001, pp. 70-77.

[35] D. M. Wahyudin, S. Heindl, A. Biffl and B. R. Schatten, "In-time project status notification for all team members in global software development as part of their work environments," In Proceeding of SOFPIT Workshop 2007, SOFPIT/ICGSE, Munich, 2007, pp. 20-25.

[36] J. D. Herbsleb and D. Moitra, "Global software development," IEEE Software, vol. 18, (2), 2001, pp. 16-20.

[37] R. Welborn and V. Kasten, The Jericho Principle, how Companies use Strategic Collaboration to Find New Sources of Value. John Wiley & Sons, Inc., Hoboken, New Jersey, 2003.

[38] H. Holmstrom, E. O. Conchuir, P. J. Ågerfalk and B. Fitzgerald, "Global software development challenges: A case study on temporal, geographical and socio-cultural distance," In Proceedings of IEEE International Conference on Global Software Engineering (ICGSE'06), IEEE, 2006, pp. 3-11.

[39] V. Casey and I. Richardson, "Virtual teams: Understanding the impact of fear," Software Process Improvement and Practice, vol. 13, (6), 2008, pp. 511-526.

[40] B. Al-Ani and D. Redmiles, "Trust in distributed teams: Support through continuous coordination," IEEE Software, vol. 26, (6), 2009, pp. 35-40.

[41] G. Borchers, "The software engineering impacts of cultural factors on multicultural software development teams," In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), IEEE, 2003, pp. 540-545.

[42] G. Hofstede, Culture's Consequences. Comparing Values, Behaviors, Institutions, and Organizations, Across Nations. Sage Publications. London, 2nd edition, 2001.

[43] K. H. Möller and D. J. Paulish, Software Metrics: A Practitioner's Guide to Improved Product Development. Institute of Electrical & Electronics Enginee, London, 1993.

[44] CMMI, "CMMI for development," Tech. Rep. version 1.2., Technical Report CMU/SEI-2006-TR-008, 2006.

[45] W. A. Shewhart, Statistical Method from the Viewpoint of Quality Control. Graduate School of Agriculture, Washington, 1939. Referenced in W.E. Deming: Out of Crisis. Cambridge, Mass.: MIT Center for Advanced Engineering Study, 1986.

[46] R. S. Kaplan and D. P. Norton, "The balanced scorecard-measures that drive performance," Harward Business Review, (No. 92105), 1992, pp. 71-79.

[47] G. Lawrie and I. Cobbold, "Third-generation balanced scorecard: Evolution of an effective strategic control tool," International Journal of Productivity and Performance Management, vol. 53, (7), 2004, pp. 611-623.

[48] D. Card, "Integrating practical software measurement and the balanced scoreboard," In Proceedings of the 27th Annual International Computer Software and Applications Conference COMPSAC 2003, 3-6 Nov. 2003, pp. 362- 363.

[49] PRISMA, Productivity in Collaborative Systems Development, ITEA project (2008-2011) number 07024, Project info page, URL: http://www.itea2.org/project/index/view/?project=237 (Accessed 19.12.2012).

[50] J. Eskeli, J. Maurolagoitia and C. Polcaro, "PSW: A framework-based tool integration solution for global collaborative software development," In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11), Barcelona, Spain, 2011, pp. 124-129.

[51] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley Pub Co, Addison-Wesley Object Technology Series, 1999.

[52] P. Kruchten, "A rational development process," CrossTalk, vol. 9, (7), 1996, pp. 11-16.

[53] E. Demirors, G. Sarmasik and O. Demirors, "The role of teamwork in software development: Microsoft case study," In Proceedings of the 23rd EUROMICRO Conference, New Frontiers of Information Technology, 1997, pp. 129-133.

[54] M. Hoegl and H. G. Gemuenden, "Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence," Organization Science, vol. 12, (4), 2001, pp. 435-449.

[55] ITEA 2, Information Technology for European Advancement, ITEA 2 homepage, URL: http://www.itea2.org/ (Accessed 19.12.2012).

[56] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: http://www.tekes.fi/eng/ (Accessed 19.12.2012).