Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 86, Issue 15, June/July 2018, Pages 1–23
https://doi.org/10.7250/csimq.2018-15.01

Using a Socio-Technical Model of a Global Software Development Project for Facilitating Risk Management and Improving the Project Structure

Ilia Bider[*], Henning Otto, and Saga Willysson

Department of Computer and System Sciences, Stockholm University, Stockholm, Sweden

ilia@dsv.su.se, h.otto@me.com, s.willysson@gmail.com

Abstract. Any global software development project needs to deal with distances – geographical, cultural, time zone, etc. – between the groups of developers engaged in the project. To successfully manage the risks caused by such distances, there is a need to explicate and present the distances in a form suitable for manual or semi-automatic analysis, the goal of which is to detect potential risks and find ways of mitigating them. The article presents a technique of modeling a global software development project suitable for such analysis. The project is modeled as a complex socio-technical system that consists of functional components connected to each other through output-input relationships. The components do not coincide with the organizational units of the project, and their teams can be distributed through the geographical and organizational landscape of the project. The modeling technique helps to explicate and represent various kinds of distances between the functional components to determine which of them constitute risk factors. The technique was developed during two case studies, of which the second is used for presenting and demonstrating the new modeling technique in the article.

Keywords: Global software development, GSD, Distance, System, Risk management, Modeling, Team, Socio-technical.

1 Introduction

It is commonly accepted that a Global Software Development (GSD) project has a number of characteristics that makes the task of managing such a project more difficult than managing a one-site project. This difficulty is connected to the various distances between the members and teams of a GSD project, which can be of different sorts: geographical, time-zone, socio-cultural, etc. [1]. These distances, when not taken into consideration, can result in decreased project effectiveness [2], [3]. To mitigate the risks of failure or underperformance due to the specific distances, there is a need to pinpoint the places in the project where the risks can materialize. One

* Corresponding author

Reference: I. Bider, H. Otto, and S. Willysson, "Using a Socio-Technical Model of a Global Software Development Project for Facilitating Risk Management and Improving the Project Structure," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 15, pp. 1–23, 2018. Available: https://doi.org/10.7250/csimq.2018-15.01

Additional information. Author ORCID iD: I. Bider – orcid.org/0000-0002-3490-6092. PII S225599221800086X. Received: 07 October 2017. Accepted: 10 June 2018. Available online: 31 July 2018.

way of doing the latter is by building a model of a GSD project that reveals these places in an explicit form.

In this article, we suggest a way of modeling a GSD project aimed at potential risk detection that is based on viewing the project as a complex system. As most important actions in such a system are completed by people, a GSD project belongs to the category of so-called socio-technical systems. A socio-technical system is considered as consisting of people organized in some way who are using certain techniques and technology to ensure functioning of the system [4]. More particularly, we suggest a way of presenting a GSD project as a number of functional components, e.g. *Requirements engineering*, *Design*, *Coding*, *Testing*, in constant interaction with each other. Components are connected to each other through output-input relationships, meaning that the outputs produced by one component are used as inputs to other components. Such functional system decomposition is often used when modeling organizations, for instance, it serves as the basis for IDEF0 notation [5]. However, output-input relationships reveal only part of the structure of a socio-technical system in general, and a GSD project in particular. A number of phenomena important for identifying risks remain outside the pure output-input relationships. To these, for instance, belong:

- The teams that complete activities in different components may intersect, e.g. some members of the requirements engineering (RE) team may participate in the design activities and/or some members of the design team may participate in the RE activities. Lack of such intersection is considered as a risk factor: see, for instance, [6].
- Functional components can work in sequence or (at least partly) in parallel.
- An output that is produced by one component may be misinterpreted by the component for which it serves as input [7]. Therefore, an additional channel of (informal) communication to clear such misunderstandings needs to be established.

In the technique for modeling GSD projects discussed in this article, we include means for presenting, in a graphical form, many issues related to the complexity of a GSD project. This would facilitate easier understanding and meaningful discussions of potential risks and their mitigation. A new project modeling technique presented in this article is based on our previous work [8] on high-level modeling of business processes.

The rest of the article is structured according to the following plan. In Section 2, we present an overview of the GSD literature to introduce the reader to the set of problems related to GSD. In Section 3, we give our research background that includes a short overview of our research methodology and the history of the research project. In Section 4, we introduce the main concepts of presenting a software project as a socio-technical system using a simplified example, analyze the risks inherent to the socio-technical system under consideration, and briefly discuss how these risks can be mitigated. In Section 5, we present our proposal on graphical presentation of a GSD project as a socio-technical system aimed at detecting the areas with potential risks. Section 6 discusses the challenges on the way to the industry adoption of our proposals and the ways of meeting these challenges. Section 7 summarizes the results achieved and discusses plans for future research.

The material in this article was first presented at the 10th IEEE International Conference on Global Software Engineering (ICGSE) and published in the proceedings of this conference [9]. The current article is a revised, restructured and extended version of [9], in which the content of Section 6 is completely new.

2 GSD Literature Overview

Standard software development processes are mostly challenged in GSD as they are subjected to a higher degree of complexity in various areas. Hence, for projects to succeed, they must be able to employ resources without concern for geographic location, attain a shared understanding among participants and effectively manage coordination issues [10]. The increased level of complexity is especially identifiable in project coordination because of differences in time zone and location, and difficulties in communication due to insufficient proximity and challenging cultural diversity. A lower frequency of communication also leads to a reduced effectiveness in communicating [10]. Challenges, furthermore, occur in cooperation where lack of trust between development teams and lack of devotion to the project can cause problems, and finally, in infrastructure management which is tasked with consolidating diverse contexts [11]. Inadequate contextual information can be a cause of misunderstandings and hinders contact initiation [10].

Software developers are forced to cooperate and bridge various types of distances, such as geographic, time-related and cultural, which together can be summarized as "global distance" [12]. Different types of distances can be identified as the main barriers in successful GSD projects, as they impair communication, coordination and control processes. The nature of distance can be classified as "temporal, geographical and socio-cultural" [1].

Time separation between distributed teams due to time zone differences is often referred to as temporal distance. Competing schedules and corporate priorities as well as different working hours or cycles further complicate this issue which leads to less overlapping time that can be used for interactive and synchronous communication [1]. Temporal distance causes an upsurge in reaction time while offering fewer chances for real-time cooperation [12].

Geographical distance is not only a measure of physical proximity but also includes the ease of visiting another agent involved in the process. It can decrease the volume and strength of communication, which, in turn, indirectly affects coordination and control mechanisms leading to decreased effectiveness [2], [3]. Moreover, geographical distance impedes the exchange of tacit knowledge as well as building of working relationships between team members because opportunities for informal meetings are missing [13].

Socio-cultural distance is a complicated measure that revolves around linguistic and political aspects, organizational and national culture, as well as personal rationale [2]. The greatest challenge inherent to this dimension is the development of a shared understanding between individuals from diverse backgrounds.

Coordination is one of the most challenging aspects of global software development and lack of it is often the reason why projects do not succeed [14]. The existence of distances has a strong impact on coordination due to its adverse effects on communication channels [3]. Geographical and temporal distances can cause communication problems, in particular, related to the rate of frequency and quality of communication [13]. For instance, tacit knowledge cannot be spread as easily through informal communication channels compared to cases where the distance does not exist or is short. This, in turn, negatively affects a mutual understanding of project aims, requirements and activities [14]. Further exacerbating this issue are socio-cultural distances, which can be the reason for misunderstandings and errors due to divergent underlying assumptions [13]. Thus, insufficient cultural recognition combined with deficient communication are often mentioned as the core challenges that project managers need to overcome in globally distributed environments [15].

The characteristics of GSD projects, listed above, require a strategy for mitigating the risks inherent in this kind of project. To mitigate the risks, they need to be discovered. Some help in discovering the risks connected to the project structure can be obtained from the empirical works that relate structural characteristics of the project organization to the economic characteristics of the product and/or project, such as modularity, quality, productivity, and profitability. Two examples of such works are summarized below.

Paper [16] investigates the relationship between the type of project organization and the level of code modularity by contrasting the products produced by commercial organizations to the ones produced by open source teams. It finds that, usually, an open source team produces code that is more modular than the ones coming from commercial organizations. Therefore, if the maintenance costs are an issue, dispersion of the project members and less face-to-face communication can be advantageous. However, such kind of settings may result in less well performing code.

Paper [17] investigates the relationships between the characteristics of the project structure and economic parameters, such as productivity, quality and profitability. To the general characteristics investigated in [17] belong geographical and time differences, the number of locations, imbalance between the teams in size, imbalance between work experience of teams, and project methodology (phase-based, iterative). The project characteristics above affect economic parameters differently. For instance, an increase in the number of sites and imbalance in the size of the teams both boost productivity at the expense of quality [17].

Though the empirical works above can help in identifying the risks, the relationships between the project characteristics and project outcomes discovered in these works are of statistical nature. They may or may not be relevant in a particular GSD project. In this work, we focus on finding a way of discovering potential risks in a particular GSD project by investigating the detailed setting of this project.

3 Research Background

The research presented in this paper belongs to the Design Science (DS) paradigm [18], [19]; in particular, we use a DS approach suggested in [20]. DS is related to finding new solutions for problems known or unknown [21]. To count as a DS solution, it should be of a generic nature, i.e. applicable not only to one unique situation, but to a class of similar situations: see Principle 1 in [22]. Another difference between DS and practical design aimed at finding a solution for a particular problem is that a DS research project does not need to produce a solution to the problem that initiated the research. While developing a solution, or artifact in terminology of [18], [19], the researchers may discover that it might suit a different problem, and switch their attention to the latter.

The research project from which we present results was started as a problem-solving project not related to GSD. The initial problem was discovered in a local practice of a small consulting company that developed a tool for building computerized support for business processes. The problem in a generic form was defined as: “given a particular business process, find a tool suitable for building computerized support for this business process”. The solution designed for this problem included a new technique for business process modeling on a high level, which was called a step-relationship model [8]. The model is defined as consisting of a number of steps and relationships between them, such as output-input relationships, parallel dependencies, etc. Relationships are defined through a set of square matrixes that can be visualized via arrows drawn between the rectangles that represent the steps.

The first practical test of the step-relationship modeling technique was completed on a GSD project at a large ICT company [23]. This test was considered as a proof of concept/demonstration in terms of DS [18], [19]. The organization in question had transitioned from a waterfall-like, phase-based software development approach to a more flexible iterative development process using the Scrum framework for project management. Building a model for the GSD project in this organization had the aim of analyzing the appropriateness of the software tools employed in the process. The model was built based on the interviews conducted with various members of the project, and investigating internal documentation.

During the final discussions about the model with the project management, it was suggested that the modeling technique we employed might be useful for other purposes than investigating the appropriateness of tools. The most promising area for new application was identified as assessment of risks connected to the distributed structure of the project and improvement of this structure. To be suited as a solution for the new problem, the step-relationship modeling needed to be further developed in two directions:

1. It had to be enhanced to represent various distances encountered in GSD projects.
2. A proper graphical visualization for the model needed to be found so that it would be easier to discuss the risks, and the ways to detect and mitigate them in the staff meetings.

A new DS research project was started with the aim to enhance the step-relationship process modeling technique to suit the goal of identifying potential risks in GSD projects. The results achieved so far are presented in Sections 4 and 5, where:

- Section 4 introduces a number of concepts needed for: (a) considering a software project as a socio-technical system, and (b) discussing the risks involved in such a project and the ways of mitigating them.
- Section 5 presents proposals of a visual modeling technique that helps to identify, discuss, and mitigate risks in a particular software development project.

4 Software Project as a Socio-technical System

4.1 Introducing the Main Concepts

To introduce the main concepts of presenting a software project as a socio-technical system, we will use a simplified example shown in the diagram of Figure 1, which, to some extent, represents a classical model of software development: see, for instance, [24]. In this diagram, the software project is presented as a system that consists of four functional components: (1) *Requirements Engineering* (RE), (2) *Design*, (3) *Coding* and (4) *Testing*. These components are related through output-input links that are represented in Figure 1 as arrows going from one component to another.

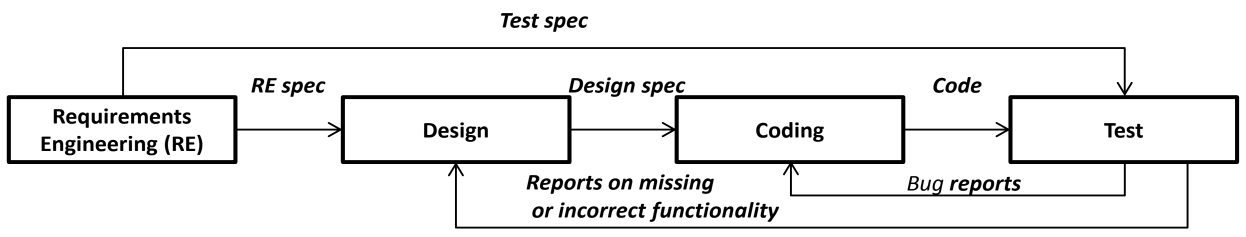

Test spec
RE spec
Design spec
Code
Requirements Engineering (RE)
Design
Coding
Test
Reports on missing or incorrect functionality
Bug reports

Figure 1. Simple model of a software project as a system (note that connections between the project and its environment are not shown)

The diagram in Figure 1 does not prescribe in which order the functional components work. The order may be “in turns”, when the “next” component idles until the full output from the “previous component” is received as its input. Alternatively, the components can work in parallel delivering their outputs in smaller portions to the next components in line. The outputs can be delivered to the next in line components in various ways. For instance, the requirements could be delivered to *Design* by *Requirements Engineering* via giving *Design* access to a requirement management system where the requirements are stored in a structured way.

A system as in Figure 1 can work satisfactorily if the outputs produced can be interpreted unambiguously by a receiving component. Even were this possible it would be difficult to achieve for a software project. The interpretation depends “on our background, education, experience, and simply from where we are standing at the moment (our responsibilities on the project)” [7]. The challenge of creating a shared understanding is expanded in global projects due to the reduced contextual information and impaired communication mechanisms [10]. To minimize the risk that the input is interpreted in a different way than intended, a negative (corrective) feedback needs to be introduced into the system. The work of such feedback is explained in Figure 2.

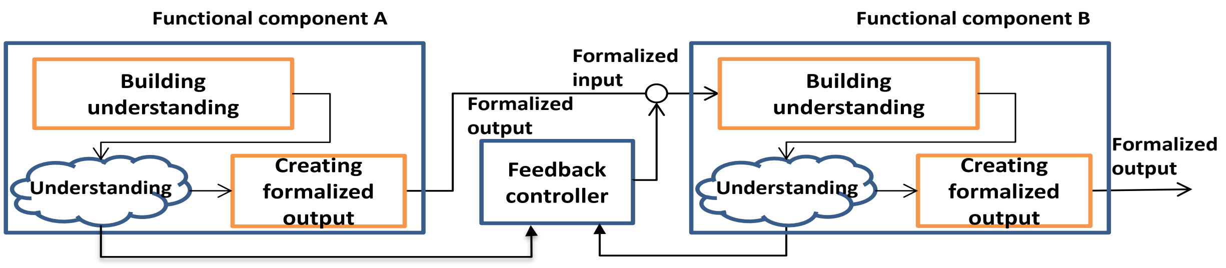


Figure 2. Negative feedback between two components of the system

Figure 2 depicts two components *A* and *B* connected through the output (or one of the outputs) of component *A* serving as input for component *B*. To create formalized output, the project members inhabiting component *A* need to build an understanding of this output in their minds, i.e. on the tacit level in the terminology of [25]; this is shown as a cloud inside the box that represents component *A*. The *Understanding* is not necessarily built before creating the *Formalized output*; this can be done in parallel or through a number of iterations. For the members of functional component *B* to create their own *Formalized output*, they need to create their own understanding (interpretation) of *Formalized input* from *A*. There is a risk that understanding *B* will not be aligned with understanding *A*, which would create a deviation possibly resulting in the production of ill-suited software by the project.

To exclude, or at least minimize, the deviation between *Understanding A* and *Understanding B*, a *Feedback controller* can be introduced into the system. As shown in Figure 2, the controller compares *Understanding A* with *Understanding B* and adds additional correcting input to *B*.

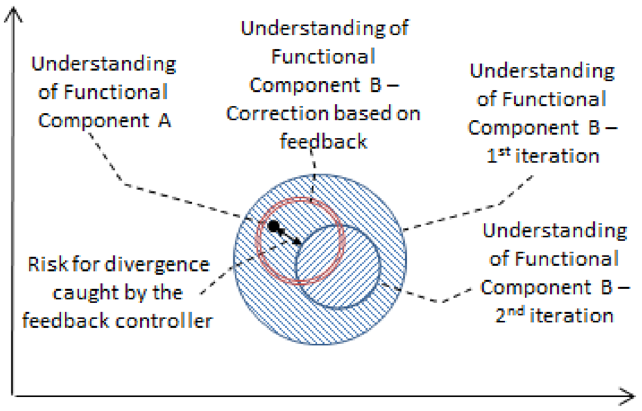
The work of *Feedback controller* is illustrated in Figure 3. The axes in Figure 3 represent the space of all possible interpretations of *Formalized input A*. This space is shown as two-dimensional for illustrative purposes; in reality, it may be multi-dimensional. *Understanding A* is represented as a point in this space. *Understanding B* is shown in Figure 3 as it develops, starting as a wider circle in the 1st iteration which in the end should be narrowed to a point, hopefully coinciding with the point which marks the position of *Understanding A* in the interpretation space.

As long as *Understanding B* "covers" *Understanding A*, as in the 1st iteration in Figure 3, there is no need for the *Feedback controller* to take any action. However, as soon as the coverage vanishes, as in the 2nd iteration in Figure 3, the *Feedback controller* reacts and provides a signal to "move" *Understanding B* in a way that it again "covers" *Understanding A*, as represented by the double line circle in Figure 3.

Naturally, in a software project, the feedback controller cannot be implemented as a mechanical, analog or digital device. It should be implemented in some other way, e.g.:

- Informal communication between the project members inhabiting functional components *A* and *B* by having a channel for questions and answers, or regular meetings. This way of arranging feedback requires that some members of the team of component *A* are available even if the task entrusted to them has been completed.
- The teams assigned to components *A* and *B* have *substantial intersection* so that intersecting parts know *Understanding A* on a tacit level, and can transfer it to the rest of the team of component *B* through socialization, in terms of Nonaka's theory of knowledge creation [26].
- Team *B* has access to the internal working documents produced by team *A*, e.g. meetings minutes, protocols of meetings with stakeholders, video recordings, etc. This can be made available as documents or as traces in a computer system that supports the work of team *A*, if such system is in use by team *A*.

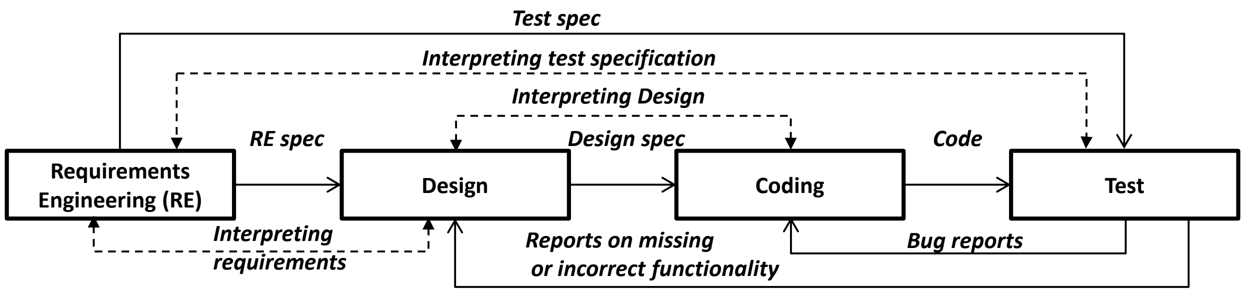
Note, also, that discovery of divergence and thus need for correction falls in the area of responsibility of Team *B*. The discovery, for instance, can happen when (a) Team *B* starts having doubts that they understand the input properly, or (b) they cannot continue to narrow down their understanding as they feel that some information is missing or incorrect, e.g. there is a contradiction.



Understanding of Functional Component A
Understanding of Functional Component B – Correction based on feedback
Understanding of Functional Component B – 1st iteration
Risk for divergence caught by the feedback controller
Understanding of Functional Component B – 2nd iteration

Figure 3. Interpretation space and the work of the feedback controller

The feedback connection between the system's components is needed for any output-input relationship that has a risk of misunderstanding. We represent this connection with a dashed double arrow connecting the functional blocks with a label of what type of information is to be used by the feedback controller. For instance, adding feedback connections to the system diagram of Figure 1 produces the system diagram of Figure 4. Note, that we have not provided feedback connections between *Test* and *Coding*, and *Test* and *Design*, regarding the inputs as unambiguous. The latter might not hold true in every case, which would require adding feedback connections between these components as well.



Test spec
Interpreting test specification
Interpreting Design
RE spec
Design spec
Code
Requirements Engineering (RE)
Design
Coding
Test
Interpreting requirements
Reports on missing or incorrect functionality
Bug reports

Figure 4. Diagram from Figure 1 complemented with feedback connections

4.2 Analyzing Risks Related to System Functioning

In the sub-sections that follow, we discuss the risk factors in the software project using the diagram in Figure 4 for illustration purposes. The factors we consider are the ones that are related to the structure of the software project and the fact that people rather than machines are completing the work in the system's functional components. In this discussion, the issues related to the use of appropriate project management methodology or deployment of people with the right qualifications and background are not considered.

4.2.1 Parallel Dependencies between Functional Components

Parallel dependency [8] exists in a case where two components *A* and *B* have an output-input relationship, and *A* works in parallel with *B* and provides inputs for *B* in portions. This, for instance, is natural for *Coding* and *Test*. It can also be the case for *RE* and *Design*, and *Design* and *Coding*. When the portions delivered are independent from each other, no major problem arises from the parallelism. If, however, the next portion contradicts or negates the previous one, the situation can become confusing. The problem arises if the previous portion received by *B* has already been processed and incorporated in the formalized output. This predicament becomes even more critical if this part of the output has already been sent to the next functional component. Then, apart from partially redoing the work, a signal should be sent to the next component(s) to invalidate part of the input they have already received. Parallel work requires very good communication channels between the functional components, so that early warnings on the possible contradiction are sent and received promptly through the chain. As parallel dependencies must be managed effectively across remote locations, it is paramount to recognize the necessary coordination activities [10].

4.2.2 Heterogeneity Inside Functional Components

Heterogeneity concerns people that belong to the team of a given functional component. It can cause conflicts and create barriers for common understanding of what is to be produced by the component. Heterogeneity has several dimensions, including:

- *Professional* – a team consists of people who belong to different professions, e.g. management, business analysts, programmers. Different professions use different terminology and see things differently, which can lead to misunderstanding and conflicts.
- *Organizational* – a team can consist of people who belong to different organizations. It is not uncommon that a large project may include several sub-contractors, each having different organizational values and goals, which can create tensions between the members of the team.
- *Geographical* – a team is distributed across different geographical locations, and time zones, which can impede collaboration.
- *Cultural* – team members can have different cultures due to their affiliation to different professions, organizations, and countries/regions. As culture affects the way people think and act, a different cultural background can create misunderstanding and conflicts.

4.2.3 Distance between Functional Components

The distance between functional components features the same dimensions as heterogeneity which was discussed in the previous sub-section. Suppose that teams for all functional components are homogeneous, i.e. all members of each team belong to the same profession, organization, location and culture. Then one of the risks to be considered is the distance between the teams of the functional components that are connected through output-input relationships. Firstly, the chance of *Understanding B* in Figure 2 diverging from *Understanding A* will increase with the distance. Secondly, communication via the feedback connection can be inefficient, for instance, due to different jargons, or cultural differences. Note that having heterogeneous teams may shorten the distance between the components in a case where both teams have intersecting mixtures of profession, cultures, etc. Hence, the ability to arrange heterogeneous teams for the purpose of cooperation inside each functional component may alleviate the problems related to the distance between the functional components.

4.2.4 Absence or Deficiency of Feedback Connections

A project may rely too heavily on the formalized outputs as inputs for the next components and does not provide means for effective functioning of the feedback controller from Figure 2. For instance, the RE team might be dissolved after completing its work, or move to another project and become unavailable. Such a situation leaves the divergence of *Understanding A* and *B* in Figure 3 uncorrected, which can result in erroneous software being produced by the project.

4.3 Mitigating Risks

As we consider a software project as a socio-technical system, the risks identified in the previous section could be mitigated in two ways: (1) through the proper arrangement of the social structure of the system, and (2) through its technical infrastructure.

The first class of measures for risk mitigation includes, for instance, organization of the project in a way that guarantees substantial *intersections* between functional components (more exactly, their teams), e.g. some members of the RE team work with design as well. This simplifies the task of the feedback controller, as some members of the functional component that produces the input are readily available for Q&A sessions needed to detect and correct possible discrepancies. A socially related measure to deal with the heterogeneity of a team is to have training sessions where the members of the team learn the each other's preferred ways regarding their professions and/or cultures. As has already been mentioned, having heterogeneous teams can help to minimize the distances between the connected functional components. Thus ensuring that the heterogeneous team collaborates properly can also help to mitigate the risk connected with distances between the teams, or at least some of them.

Measures regarding creating appropriate technical infrastructure include, for instance, introducing groupware type tools for each functional component. Hence, all intermediate decisions, notes, documents, etc. would be stored and could be made available to the team that will work with the output of the given functional component. Access to such internal material could help to organize the work of the feedback controller; even in the situation of limited access of team *B* to members of team *A* in Figure 2. The latter can occur, for instance, if team *A* left the project, or if there is a substantial time zone distance between the functional components *A* and *B*. Using the traces of the internal work of team *A* requires team *B* to understand the working ways of team *A*. Thus, bridging the professional distance between the functional components *A* and *B* is necessary. An alternative solution here would be having a tool that can translate the information from the internal systems used by *A* to the language understandable for team *B*.

Although it is a important practical issue, the ways of reducing the risks are not the focus of this article [†]. Our goal here is to show how considering a software project as a socio-technical system as in Figure 4 could help to identify potential risks that exist in a particular project. This is accomplished in the following sections via using an example from practice.

5 Designing a Graphical Model for Identifying Risks

5.1 Background and Goal

As we have argued in the previous sections, identifying and mitigating risks in a software project needs to take into consideration socio-technical dependences between the functional components of the software project. To explicate these dependencies to the diverse people involved in the project, e.g. management, requirements engineers, developers and testers, their visual representation could be of great help. Several approaches to visualizing socio-technical dependencies have been suggested. Mostly these approaches are based on the idea of designing a tool that can extract and visualize such dependencies from information that can be obtained from the tools that support software development, such as configuration and version management systems, bug reports management systems, etc. A typical work in this area is [27], which presents a tool, called Ariande, for visualizing socio-technical dependencies. Ariande is a plugin for Eclipse that includes functionality for analysis of technical dependencies between the components of a software system. Based on authorship information, Ariande then "translates" the technical dependencies between the components into social dependencies between the developers engaged in the project. Ariande has a

[†] The issues related to the tools to be used to minimize the risks are discussed [8].

goal of exploring these relationships to enable coordination of global software developers through visualization of these dependencies. Other tools of similar sorts are overviewed in the survey presented in [28].

Our goal with graphical presentation of a socio-technical model of a GSD project differs from the goal of the works cited above. These works concentrate on visualizing (a) specific dependencies (b) based on the information that can be automatically extracted from the tools that support software development. Our goal is different, namely:

- To visualize more abstract properties of the software projects, like distances inside and between various teams, in order to localize and mitigate the risks related to these distances.
- Do it independently whether or not it is possible to extract the information from the tools used in the project.

More exactly, our goal is to visualize the concepts of a software project as a socio-technical system introduced in Section 4. The idea is to find appropriate graphical means of presenting these concepts, so that the model will be easy to analyze by human beings, e.g. during a meeting.

All examples presented in this section are related to the project that we investigated in the business case described in the next sub-section. The resulting graphical model consists of a number of views, each of them representing a combination of the concepts introduced in Section 4.1. These views are discussed in detail in the following sub-sections, starting with Section 5.3.

5.2 Business Case and Knowledge Base

The business case organization used as an example in this paper is a multinational ICT corporation that develops a large variety of software solutions for telecommunications technology. One of the software maintenance departments at the case organization, and its respective software development process, was selected for our investigation. The process exhibits several characteristics that are inherent to globally distributed settings. Firstly, four different locations are involved in the software development process. The sites are distributed across four different countries, of which three are located in Europe, and one resides in Asia. Secondly, a number of different organizations collaborate in the software development. These organizations can be categorized, according to their type, as the head office, semi-independent sub-division and independent sub-contractor. Thirdly, people with diverse professional backgrounds are engaged in the process. In particular, the groups of professions include the operational management, technical design staff and test engineers. The project employs Scrum as its development methodology; the developers work in parallel and use short iteration cycles and continuous integration.

As the selected case represents a typical large software development project in a global setting, it offers sufficient challenges, and provides an adequate test scenario for building a socio-technical model, and also for designing extensions to the step-relationship modeling technique from [8] to make it suitable for this end.

The main bulk of information for building a graphical model of the project was obtained via interviewing the key people in the project, i.e. a project manager placed in the head office and three sub-project managers, one for each respective geographical location. Additionally, internal documentation has been studied. The same people were used to verify the model and assess the enhancements to the step-relationship modeling technique that were added when developing the graphical model. This was done in an iterative manner during brainstorming sessions.

Besides the information obtained from the key participants of the GSD project, the following three sources of knowledge were used when designing the extensions to the step-relationship model:

1. The existing body of knowledge on characteristics of GSD projects, see the literature overview in Section 2.

2. The existing body of knowledge related to visualization of dependencies in software projects, such as [27], [28], as well as more general recommendations related to visualization, such as [29].
3. Own experience of software development.

5.3 The Main View – Output-input Relationships

The main view of the model is the output-input view shown in Figure 5. In this view, rectangles represent functional components of a socio-technical system, while solid line arrows represent output-input relationships between the components. A label inside a rectangle describes the function completed by the component; a label on an arrow identifies what kind of an output is sent from one component to another. Dashed arrows represent feedback connections as defined in Section 4.1. In Figure 5, we assume that each solid arrow (output-input relationship) is accompanied by a feedback connection that is not shown on the diagram. The dashed arrows are explicitly present only if corrective signals are coming from the component not directly connected to the given one: see, for instance, the dashed arrow between *Application Testing* and a *Feature X Development*.

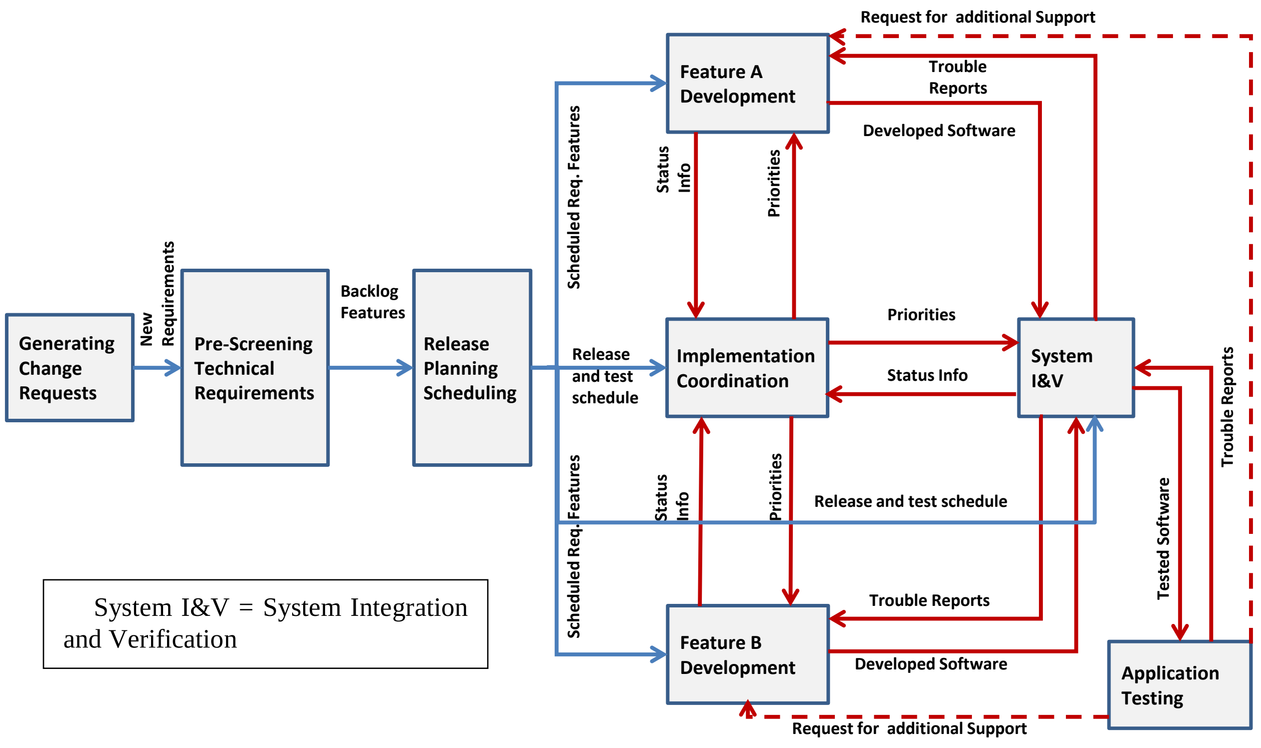


Figure 5. System view with output-input relationships

5.4 Visualizing Parallel Dependencies

As described in Section 4.2.1, a parallel dependency exists between two functional components if they are connected via an output-input relationship, and at the same time, they can work, partially or fully, in parallel. We use two approaches to representing parallel dependency in a graphical model. The first one is quite simple, i.e. via using red color to mark output-input relationships between the components that have parallel dependencies. This is depicted in Figure 5, which actually is a view that combines both output-input relationships and parallel dependencies.

A more elaborate way of depicting parallel dependencies is presented in Figure 6. The diagram in Figure 6, called timeline intensity diagram, plots the process components on a timeline running from left to right and indicating the passing of time. The components that are depicted next to each other are executed in a sequential manner, whereas the components that are stacked on top of each other are executed simultaneously. This notation provides a detailed view of time dependencies, showing partial as well as full overlapping. Furthermore, the diagram shows the workload intensity, and how it changes during the execution. For this, the vertical dimension of the shape that depicts a functional component is used. The greater the height of the shape the more work is required for the respective process component at a particular point in time.

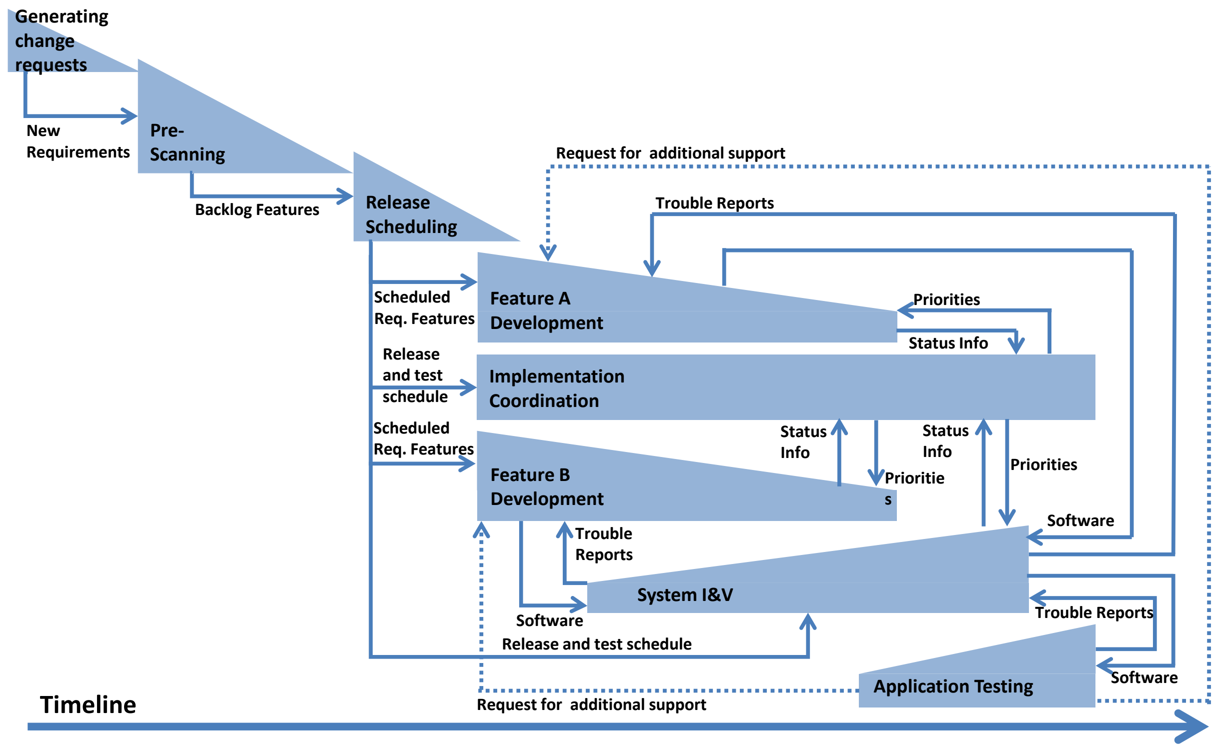


Figure 6. Timeline intensity for parallel dependencies

The diagram in Figure 6 shows, in an explicit way, all parallel dependencies that exist in the project, i.e. functional components that run in parallel while providing inputs to each other. For such dependencies, additional attention is necessary in terms of coordination efforts to ensure smooth process execution.

5.5 Visualizing Teams' Intersections

As discussed in Section 4.3, intersecting teams is one of the ways of mitigating risks inherent to a complex project, thus they need to be visualized in the model. Currently, intersecting teams are visualized indirectly via showing the composition of each functional component in terms of organizational structure, as explained below.

From the organizational structure point of view, several organizational units can participate in the project. Each functional component falls into the area of responsibility of a certain unit. This, however, does not imply that the team of the given functional component consists of the members of only one unit. For some functional components, their teams consist of members from several units, which points to intersection between the teams. The organizational units engaged in our business case and the distribution of responsibilities between them are presented in Table 1.

Table 1. Mapping of functional components and team responsibility

Functional Component	**Responsible Unit (in bold)** + other participants
Generating change requests	**Others** (not part of the project under consideration)
Pre-Screening	**System Management** + representatives from feature teams
Release Planning Scheduling	**Project Management** + sub-project managers from feature teams
Feature A Development	**Feature development – Country 2**
Feature B Development	**Feature development – Country 3**
Implementation/ Coordination	**Project Management** + sub-project managers from feature teams and System I&V
System I&V (Integration & Verification)	**System I&V** (Integration & Verification) + representatives from feature teams
Application Testing	**Others** (not part of the project under consideration)

Visualization of responsibility for, and composition of, each functional component is achieved via assigning two colors to each functional unit, border color and a filling color, as done in Figure 7. The color assignments for our business case project are presented in Figure 7 under *Notation* in the left bottom corner. The border of a functional component in Figure 7 is colored in accordance with the border color of the unit that is responsible for this component. In addition, each component in the diagram is sliced horizontally into sectors to represent different units that contribute to its work. Each sector receives the filling color of the corresponding unit, the height of the sector indicating the relative proportion of the participants from this unit.

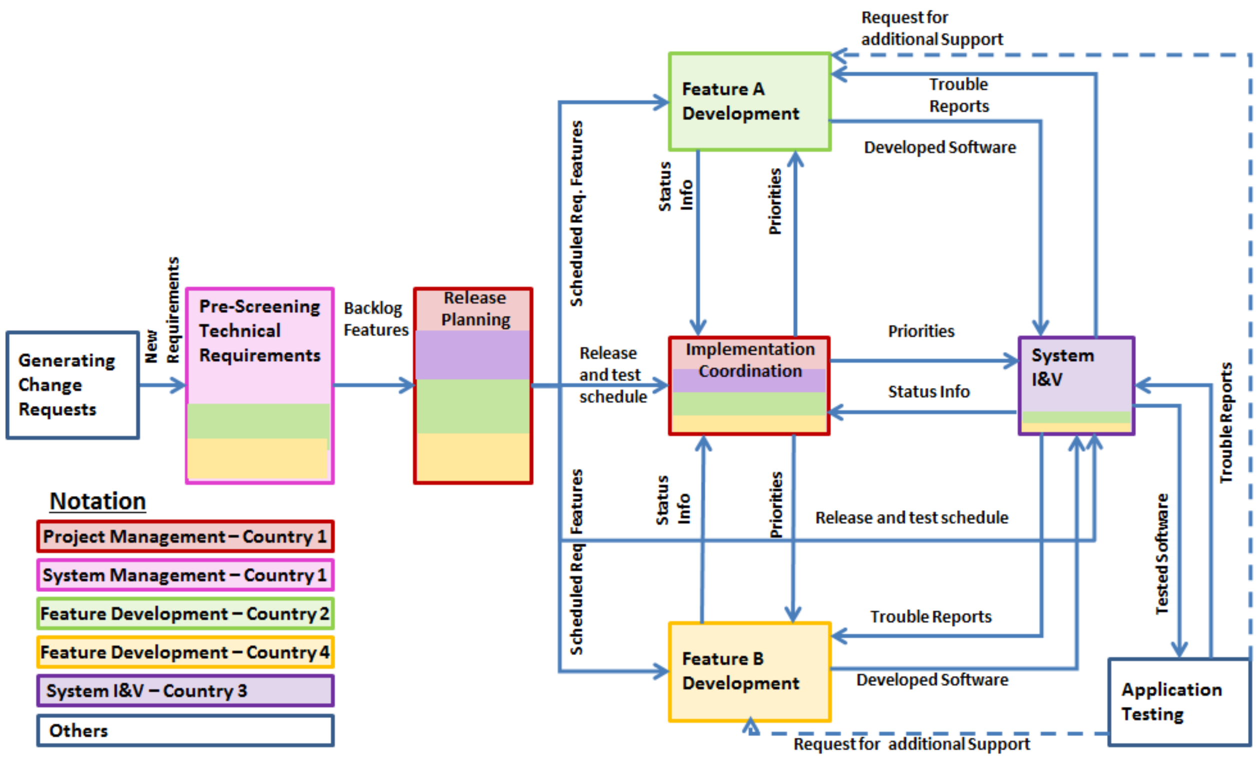


Figure 7. Teams' composition and intersection

Actually, Figure 7 represents a combination of the output-input view with team composition. Indirectly, this diagram also shows component teams' intersection. If two components have the same filling color inside, e.g. *Feature A Development* and *Implementation Coordination*, their teams intersect. This can be used for identifying intersecting teams between the components that have output-input relationships.

An advantage of using team composition for identifying intersecting teams is that a diagram like the one in Figure 7 is relatively easy to draw, even manually. Nevertheless, such indirect visualization has its drawbacks, as it does not always show the size of an intersection. For instance, the composition of *Implementation Coordination* gives an idea of the size of intersection *Feature A Development* and *Implementation Coordination* in relation to the size of the *Implementation Coordination* team. However, it gives no idea of the intersection size in relation to the size of the *Feature A Development* team. In absolute numbers this intersection is quite small – just one team project manager. In relation to the size of the *Implementation Coordination* team, this is a sizable intersection, but in in relation to the size of *Feature A Development* – it is not. Other possible ways of presenting intersecting teams are discussed in Section 6.

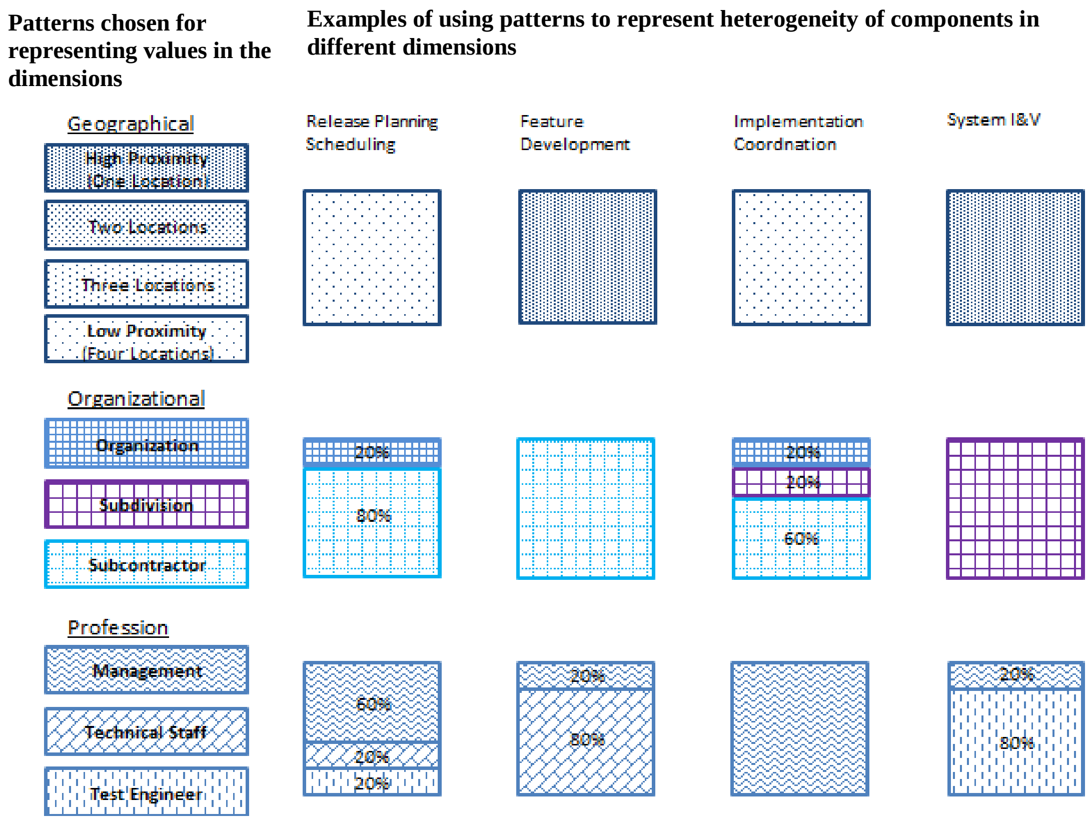
5.6 Visualizing Components Heterogeneity

We have chosen two approaches to visualization of components' heterogeneity introduced in Section 4.2.2. Both approaches are illustrated in Figure 8 where three dimensions of heterogeneity – geographical, organizational, and professional – are presented. The first approach represents heterogeneity by the density of dots included in the background pattern of a functional component. The high density of dots in the pattern indicates homogeneity of the component, the low density of dots points to the heterogeneity of the component in the given dimension. In Figure 8, this approach is applied so as to represent the geographical dimension of heterogeneity. The high density of dots means geographical proximity, i.e. all members of the team are in the same location (a homogeneous team in the geographical sense), whereas a low density of dots symbolizes a geographically dispersed team.

An advantage of the density approach of the visualization is that it gives an overall impression on the level of heterogeneity. Another advantage is that the level of heterogeneity in this case is easy to calculate. The drawback is that the picture can be misleading. For instance, the visualization will be the same when we have 50 % of team members from one country and 50 % from another, and when we have 90 % from one country and 10 % from the other.

The second approach for visualizing the level of heterogeneity is by using distinct patterns to identify the different values of the dimension in which heterogeneity is to be presented. In Figure 8, this approach is used for representing organizational and professional heterogeneity. Patterns are used to slice the rectangle that represents a given component horizontally in proportion to the percentage of team members affiliated with each pattern. Each value, in this case, is represented by a different pattern, while colors can be also used to underline the distinction. The advantage of this kind of visualization, compared with using the density of dots, is that it clearly shows the proportion of each value in the total combination. However, such visualization demands more work on assessing the proportion and drawing the diagram when the drawing is done by hand (without using a tool).

The slicing approach to visualization works well when one member of the team can be affiliated only to one value in the given dimension. This is, for instance, the case with the organizational dimension in Figure 8. When this assumption is not true, the slicing approach will be insufficient as it is practically impossible to show intersections when using sliced patterns. An example where multiple affiliations may exist is the professional dimension in Figure 8. To use the slicing visualization in this case, we made a simplification, relating a person to a profession that closely corresponds to the current position in the project. For instance, a project manager was considered as belonging to the management profession, independently of whether he/she possessed technical skills due to his/her past engagements. To represent situations with some team members having multiple affiliations, something like Venn diagrams could be used. This however is difficult to employ when the model is drawn manually. There is a need to have a tool that can help in drawing diagrams based on the numerical data supplied to it. This issue is discussed in more details in Section 6.


Patterns chosen for representing values in the dimensions
Examples of using patterns to represent heterogeneity of components in different dimensions
Geographical
High Proximity (One Location)
Two Locations
Three Locations
Low Proximity (Four Locations)
Organizational
Organization
Subdivision
Subcontractor
Profession
Management
Technical Staff
Test Engineer
Release Planning Scheduling
Feature Development
Implementation Coordnation
System I&V
20%
80%
20%
20%
60%
60%
20%
20%
20%
80%
20%
80%

Note: Pattern *Organization* in the organizational dimension represents the head office of the corporation.

Figure 8. Dimensions of heterogeneity of functional components

Note that the pattern-based visualization shown in Figure 8 can be combined with the output-input view of Figure 5 by using sliced filling of the rectangles, in the same way as it has been done for team composition in Figure 7. In this case, we get two new views that show different kinds of heterogeneity with respect to inter-team communication.

5.7 Visualizing Distances between Functional Components

The only way we have used so far for representing distances between functional components (see Section 4.2.3) is illustrated in Figure 9. Diamonds are introduced on the arrows that represent output-input relationships between the components. The density of the dots in the filling patterns of the diamonds is used for representing the distances. Figure 9 illustrates representing geographical distances between the components: the lower the density of the dots, the greater the geographical distance. The distance is measured between the units responsible for each component. Connections with greater distances need special consideration, for instance, when deciding on tools to be used for communication between the component teams. It is also possible to apply the notation from Figure 9 to the diagram in Figure 6, which makes it possible to study both distances and team intersections, since having intersecting teams may bridge some of the distances (see the discussion in Section 4.3)

The diamond notation can be used for representing other types of distances, e.g. professional or cultural. One advantage of this way of visualizing distances is its simplicity. The diamond notation can be used even when the model is drawn by hand, without using any special tool. However, this visualization can be misleading, as the distance is considered between the responsible units without taking into account the composition of the teams. A more appropriate way of measuring the distances between the teams could be by getting an average of the distances between each pair of people, one from each team. In this case, intersection of teams will automatically “shorten” the distance. For more detailed discussion of this topic see Section 6.

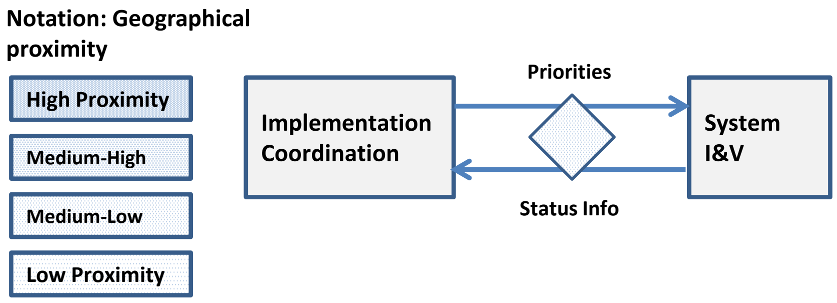


Figure 9. Geographical distances between functional components

5.8 Assessing the Usefulness

As has been already mentioned in Section 5.2, building the model for our business case was based on interviewing project participants, especially those in charge of the GSD project and its sub-projects (managers). The final model consisted of combinations of various views discussed in the previous sub-sections. Additionally to concepts discussed above, some views included information on communication/collaboration tools used inside each component, as well as between components. These were added to facilitate analysis of whether the tools employed were appropriate, considering the heterogeneity of the teams, and the distances between them.

The final model was presented both to, and discussed with, people inside the project, and managers at the higher level of organizational hierarchy. The focus of the presentations was on identifying potential risks, thus when presenting the model, we put red circles on the places that could constitute risks.

The model was presented and discussed separately with:

1. The project manager who has total responsibility for the project. This was done during several face-to-face meetings.
2. The project group that, beside the project manager, included sub-project managers and technical staff from *Feature teams* and *System Integration & Verification*. This was done during a virtual audio meeting while sharing the computer screen.
3. A management group that, beside the project manager, included various program managers. This was done during a face-to-face meeting.

The most positive feedback was provided by the project manager. This is understandable, as she was involved in the modeling project more than any other project member. The project manager saw the model being useful for project planning, not only in the current GSD project, but also in other projects in the company. In her view, when a new project is to be created, its model could be built and analyzed before actually starting the project. Another potential usage could be in assessing the tools used inside and between the components, e.g. to check whether the tools would be suitable considering the heterogeneity of the teams and distances between them.

The feedback from the project group was less definite, which can be explained by the fact that the participants of the meeting, who were not involved in model building, had difficulty in understanding the short presentation. In addition, the meeting, being virtual and without video, made it difficult to hold the discussion. Due to the geographical distances and busy schedule of the project team, it was difficult to get feedback after the meeting. Anyway, no negative comments were forwarded to us, while some members considered the model as potentially useful for (a) initial planning, and (b) introducing new members to the project.

The general attitude was more positive during the presentation to the management group. The model was considered as useful for the intended purpose – identifying potential risks. The attendees also found it interesting and useful to have a model whose basic elements *did not coincide* with the organizational structure. It gives a proper basis for discussing the problems and solutions related to communication and collaboration inside the project. A natural question arose of whether the identified potential risks actually occurred in reality. We could not give an answer to this question, as it required further research.

Besides testing the usefulness of our model as a whole, we also evaluated whether the types of visualization chosen for each view were satisfactory. Only the project manager was available for this evaluation. During the evaluation, we went through all model views, and discussed whether the chosen ways of visualizing the concepts, relevant to the given view, were satisfactory. Positive reaction has been obtained for all views. The validation, however, can be considered only as preliminary, because it included only one person, and this person was part of our modeling project. A more thorough investigation of the chosen ways of visualization is required: one that includes more people who are external to the research project.

6 Challenges on the Way to Industry Adoption

Design science research cannot generate, on its own, enough test cases for statistical validation of an artifact/solution [20]. This can only be done by the industry adopting the solution. The version of modeling technique presented in the previous sections is suitable for a research project, but it is not mature enough for the adoption. There are some challenges to be overcome in order to make the technique more attractive for adoption.

The first challenge is connected to the fact that drawing a socio-technical model is quite time-consuming, considering the multitude of views. While drawing the output-input diagram is not too difficult, producing all other views manually is quite a tiresome task. The second challenge is connected to the fact that analysis of risks based on the model is done manually. To make adoption easier, these challenges need to be addressed by automating, at least partially, both building the model, and identifying the risks.

For automation purposes, we will be using the matrix form suggested in [8] for presenting various views of socio-technical model (see Table 2).

Table 2. A fragment of the output-input matrix for Figure 5

	Release Planning	**Feature A development**	**Feature B development**	**Implementation Coordination**	**System I&V**
Release Planning					
Feature A development	Schedule Req. Features			Priorities	Trouble Reports
Feature B development	Schedule Req. Features			Priorities	Trouble Reports
Implementation Coordination	Release and test schedule	Status Info	Status Info		Status Info
System I&V	Release and test schedule	Software	Software	Priorities	

According to this form, each view is presented as a square matrix, where both rows and columns represent the system's functional components. A diagonal cell, if used, characterizes the respective component, while a non-diagonal cell characterizes relationships between the column component and row component. For instance, the output-input diagram in Figure 5 can be represented as a matrix in which a cell (*a*,*b*), where *a* is a column component and *b* is a row component‡, contains a list of formalized outputs from component *a* that serve as formalized inputs to component *b*. A fragment of such a matrix for the model in Figure 5 is presented in Table 2. Matrices of the type illustrated in Table 2 can be analyzed by formal means and even combined with each other to produce more complex views of the system that represent various combinations of perspectives.

In the rest of this section, we present examples of using our socio-technical model in matrix form to meet the challenges identified above.

6.1 Calculating Team Intersection

Basic information on teams' intersection can be presented as a matrix where the diagonal shows the size of each team, while a non-diagonal cell (*a*,*b*) shows the size of intersection between the team of column component *a* and the team of row component *b*. An example of such a matrix is presented in Table 3. This basic team intersection matrix is symmetric as the values (*a*,*b*) and (*b*,*a*) are the same. The absolute values, as in Table 3, actually, do not represent the strength of the linkage between the two components, as the strength depends on the size of the teams. To represent the strength, a derived matrix, called relative intersection matrix, can be produced by dividing the value in each cell (*a*,*b*) by the value in cell (*a*,*a*); see Table 4 that is derived from Table 3. The result, the value in cell (*a*,*b*), will show the percentage of the size of team *a* which is also engaged in the activities of team *b*. The higher the value, the tighter is the coupling between components *a* and *b*, in the sense that component *b* knows enough of what is happening in component *a* on the tacit level.

Table 3. An example of the basic team intersection matrix

	Release Planning	**Feature A development**	**Feature B development**	**Implementation Coordination**	**System I&V**
Release Planning	6	2	2	4	2
Feature A development	2	25	0	1	0
Feature B development	2	0	15	1	0
Implementation Coordination	4	1	1	4	1
System I&V	2	0	0	1	20

Table 4. A relative team intersection matrix derived from Table 3 and Table 2

	Release Planning	**Feature A development**	**Feature B development**	**Implementation Coordination**	**System I&V**
Release Planning		8 %	13 %	100 %	10 %
Feature A development	33 %		0 %	25 %	0 %
Feature B development	33 %	0 %		25 %	0 %
Implementation Coordination	66 %	4 %	7 %		5 %
System I&V	33 %	0 %	0 %	25 %	

‡ We follow a rather unusual notation for identifying a matrix cell where the first component is the column, and the second is the row to follow the direction of the flow from output (column) to input (row).

Note that the derived matrix becomes asymmetric; a tight coupling between *a* and *b* is not automatically translated into a tight coupling between *b* and *a* when the sizes of the teams are not equal.

In addition to showing percentage of intersection, Table 4 has red borders around cells that are not empty in the output-input matrix from Table 3. Formally, this means that the matrix in Table 4 is a merger of the derived team intersection matrix and simplified output-input matrix, where simplification consists of reducing the values in the nonempty cells to Boolean yes/no. The essential level of intersection in the cells with red borders indicates that the feedback controller between the cells might be ensured via the intersection. In case the intersection does not exist or is a low value, say in single digit numbers, there might be a need to implement the feedback controller via other means, e.g. communication channels, or shared space available for both teams. Such situations are marked via the red font color in Table 4.

6.2 Calculating Professional Distance between Teams

As has already been noted, the same person may possess knowledge and experience from several professions. This should be taken into consideration when measuring the professional distance. In this sub-section, we will show how the professional distance between the teams can be identified and assessed, while using the matrix form. As in the previous example, we first create a basic matrix of inter-professional relationships in the following manner. The diagonal of the matrix defines the sizes of the teams. A non-diagonal cell (*a*,*b*), where *a* is a column and *b* is a row, shows the number of members of team *a* that have practical experience of work done in component *b*. An example of such a matrix is presented in Table 5.

Table 5. An example of the basic inter-professional matrix

	Release Planning	**Feature A development**	**Feature B development**	**Implementation Coordination**	**System I&V**
Release Planning	6	4	4	4	2
Feature A development	4	25	15	2	5
Feature B development	4	25	15	2	5
Implementation Coordination	4	3	2	4	3
System I&V	3	5	4	2	20

Table 6. A relative inter-professional matrix

	Release Planning	**Feature A development**	**Feature B development**	**Implementation Coordination**	**System I&V**
Release Planning		16 %	13 %	100 %	10 %
Feature A development	66 %		100 %	50 %	25 %
Feature B development	66 %	100 %		50 %	0 %
Implementation Coordination	66 %	12 %	13 %		15 %
System I&V	50 %	20 %	26 %	50 %	

To show the strength of inter-professional connections, a derived matrix, called relative inter-professional matrix, can be obtained by dividing the value in each cell (*a*,*b*) by the value in cell (*a*,*a*): see Table 6 that is derived from Table 5. The result, the value in cell (*a*,*b*), will show the percentage of team *a* having experience of participating in activities conducted in component *b*. The higher the value, the tighter is the inter-professional connection between components *a* and *b*, in the sense that team *a* understands the nature of activities in component *b*.

Note that the derived matrix becomes asymmetric, a tight coupling between *a* and *b* is not automatically translated into a tight inter-professional connection between *b* and *a* when the sizes of the teams are not equal.

A high value of inter-professional connection between components *a* and *b* is desirable when *a* produces a formalized output to serve as input for *b*. In Table 6, such a situation is marked by cell (*a*,*b*) having a red border. Firstly, inter-professional connections can help to make the formalized output more understandable for component *b*; secondly it makes it easier to create a feedback controller between *a* and *b*, based on answering the questions from *b* to *a*, as *a* has enough people that understand the language and concerns of team *b*.

A high value of the reversed inter-professional connection between components *a* and *b* can also be helpful when *a* produces formalized output to serve as input for *b*. This value is represented in cell (*b*,*a*) which is symmetrical to cell (*a*,*b*). The high value in cell (*b*,*a*) can compensate for the low value in cell (*a*,*b*) in two ways. Firstly, a substantial number of members of *b*, having tacit understanding of activities that happened in *a*, can help team *b* to understand the formalized output from *a*, even if it has been written in the professional language of component *a*. Secondly, *b* will have enough people to arrange a feedback controller between *b* and *a*. The controller can be arranged based on the people from *b*, who understand activities and input from *a*, communicating with team *a* in the questions-answers manner. Alternatively, it can be arranged by these people from *b*, having access to the internal documentation/systems of team *a*, to look for answers themselves.

Summarizing the deliberation above, when team *a* produces input for team *b*, tight inter-professional connection either between *a* and *b* (i.e. *a* understanding *b*) or between *b* and *a* (*b* understanding *a*) will be of help in creating a feedback controller between *b* and *a*. However, the method of organizing the controller will differ dependent on which connection is stronger.

6.3 Summary

In the previous sub-sections, we presented two examples of using matrixes for defining views of the socio-technical model of a software project. In both examples, first, a basic matrix is created, based on the factual data from the project. Then, a derived matrix is built, based on a simple procedure that can be fully automated. In addition, the derived matrix is merged with another matrix also according to a simple algorithmic procedure. Based on the combined matrix, risks are analyzed and pointed out. The results of operations on the matrixes can then be converted into a graphical model by amending the basic input-output diagram of the type shown in Figure 5. The distances identified in a matrix can be represented in a form similar to Figure 9, or in a more elaborate form. This procedure can be fully automated, provided that the basic input-output diagram has been built manually.

Summarizing the above, it seems possible to automate, at least partially, the building of a socio-technical model of a GSD project in the following manner. Firstly, a basic output-input diagram is built. Secondly, a number of basic matrixes are filled. Thirdly, derived matrixes are produced via a fully automated procedure, and risks are identified through highlighting specific cells. Fourthly, based on the basic output-input diagram and derived matrixes, a number of additional graphical views are built and risks related to them, if any, are graphically highlighted. To make such automation possible, a tool needs to be built that facilitates drawing a basic input-output diagram and filling basic matrixes. Building such a tool can help to meet the challenges to adoption of our proposals by the industry.

7 Conclusion – Main Contributions and Future Plans

7.1 Contributions

According to the literature overviewed in Section 2, effects of various types of distances between the members of a GSD team have been known for some time. Various authors have suggested pragmatic ways of dealing with these effects. However, to the best of our knowledge, the goal of building a model of a software project that visualizes these distances and helps to identify the areas at risk has not, so far, been discussed in the scientific literature. Defining such a goal represents the first contribution of this paper to the theory and practice of GSD.

In Section 4, we suggested the concepts that need to be represented in a model of GSD team; some of them, like the feedback controller, not having been much discussed in the contemporary GSD literature. The choice of the concepts was made having the goal of identifying potential areas of risk. The conceptual model of a GSD project, informally presented in Section 4, represents the second contribution of this paper.

In Section 5, we presented a practical version of implementation of the conceptual model from Section 4, which we consider as the third contribution of this paper. This implementation has been tested in one case study and partly validated in that setting. Our implementation of the conceptual model from Section 4 needs further testing and improvement to meet challenges on the way to industrial adoption discussed in Section 6. However, even in its current form, this model can serve as a proof that the concepts introduced in Section 4 can serve as a basis for creating a practical socio-technical model of a GSD project aimed at risk identification and mitigation. In addition, Section 6 includes suggestions as to how the identified challenges can be met.

7.2 Plans for the Future

Our plans for the future include further development of the model along the lines presented in Section 6, and testing it in new business cases. Another area for future research could be devising measures to mitigate the risks both in the social and technical plans. This topic is also included in our plans for the future.

The model presented in the paper was designed with a view to identifying and mitigating risks in GSD projects. However, the model might be useful for other purposes, for instance, it could be used in educational activities for software engineers and managers. Investigation of areas, besides risk identification and mitigation, where our socio-technical model can be of use is also included in our plans for the future.

Acknowledgements

The authors are grateful to Thanos Karapantelakis who participated in building the model for the business case presented in this paper. We are also thankful to all members of the GSD project we investigated for spending their time while answering our questions and attending the meetings. Here, we are especially grateful to Bogumila Rutkowska, Margarita Ilyushina, Edo Zulim and Padmaja Roy Kondapalli. We are also grateful to the anonymous reviewers who provided valuable comments that helped to improve the text.

References

[1] H. Holmstrom, E. Ó. Conchúir, P. J. Ågerfalk and B. Fitzgerald, "Global Software Development Challenges : A Case Study on Temporal, Geographical and Socio-Cultural Distance," in *International Conference on Global Software Engineering*, 2006. Available: https://doi.org/10.1109/icgse.2006.261210

[2] S. Beecham, N. Carroll, J. Noll and T. I. S. E. R. C. Lero, "A Decision Support System for Global Team Management: Expert Evaluation," in *Seventh International Conference on Global Software Engineering Workshops*, 2012. Available: https://doi.org/10.1109/icgsew.2012.14

[3] D. Šmite and J. Borzovs, "A framework for overcoming supplier related threats in global projects," in *Software Process Improvement, LNCS Vol. 4257*, 2006. Available: https://doi.org/10.1007/11908562_6

[4] M. Mumford, "The story of socio-technical design: reflections on its successes, failures and potential," *Information Systems Journal*, vol. 16, no. 4, p. 317–342, 2006. Available: https://doi.org/10.1111/j.1365-2575.2006.00221.x

[5] NIST, "Integration definition for function modeling (IDEF0), Draft Federal Information Processing Standards, Publication 183, 1993," [Online]. Available: www.idef.com/downloads/pdf/idef0.pdf. [Accessed 1. 2. 2015.].

[6] E. Bjarnason, K. Wnuk and Regnell B., "Requirements are slipping through the gaps -- A case study on causes & effects of communication gaps in large-scale software development," in *21st IEEE International Requirements Engineering Conference (RE)*, Trento, 2011. Available: https://doi.org/10.1109/re.2011.6051639

[7] D. Jacobs, "Requirements Engineering so Things Don't Get Ugly," in *ICSE 2007 Companion*, Minneapolis, MN, US, 2007. Available: https://doi.org/10.1109/icsecompanion.2007.60

[8] I. Bider and E. Perjons, "Design science in action: developing a modeling technique for eliciting requirements on business process management (BPM) tools," *Software & Systems Modeling,* vol. 14, no. 3, pp. 1159-1188, 2015. Available: https://doi.org/10.1007/s10270-014-0412-6

[9] I. Bider and H. Otto, "Modeling a Global Software Development Project as a Complex Socio-Technical System to Facilitate Risk Management and Improve the Project Structure," in *Proceedings of the 10th IEEE International Conference on Global Software Engineering, forthcoming*, Ciudad Real, Spain, 2015. Available: https://doi.org/10.1109/icgse.2015.13

[10] J. D. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination," in *Future of Software Engineering (FOSE 2007)*, 2007. Available: https://doi.org/10.1109/fose.2007.11

[11] S. Abufardeh and K. Magel, "The Impact of Global Software Cultural and Linguistic Aspects on Global Software Development Process (GSD): Issues and Challenges," in *4th International Conference on New Trends in Information Science and Service Science (NISS)*, Gyeongju, South Korea, 2010.

[12] K. Ehrlich, M. Helander, G. Valetto, S. Davies and C. Williams, "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development," in *1st International Workshop on Socio-Technical Congruence*, 2008.

[13] M. Jiménez, M. Piattini and A. Vizcaíno, " Challenges and improvements in distributed software development: A systematic review," *Adv. Softw. Eng.*, vol. 2009, pp. 1-14, 2009. Available: https://doi.org/10.1155/2009/710971

[14] B. J. Noll, S. Beecham and I. Richardson, "Global Software Development and Collaboration: Barriers and Solutions," *ACM Inroads,* vol. 1, no. 3, pp. 66-78, 2010. Available: https://doi.org/10.1145/1835428.1835445

[15] M. Niazi, S. Mahmood, M. Alshayeb, M. R. Riaz, K. Faisal and N. Cerpa, "Challenges of Project Management in Global Software Development : Initial Results," in *Science and Information Conference*, 2013.

[16] I. MacCormack, J. Rusnak and C. Baldwin, "Exploring the Duality between Product and Organizational Architectures: A Test of the “Mirroring” Hypothesis," 2011. Available: https://doi.org/10.2139/ssrn.1104745

[17] N. Ramasubbu, M. Cataldo, R. K. Balan and J. D. Herbsleb, "Configuring Global Software Teams: A Multi-Company Analysis of Project Productivity, Quality, and Profits," in *Proceedings of ICSE’11*, Waikiki, Honolulu, HI, USA, 2011. Available: https://doi.org/10.1145/1985793.1985830

[18] A. R. Hevner, S. March and P. J. and, "Design Science in Information Systems Research," *MIS Quarterly,* vol. 28, no. 1, pp. 75-105, 2004. Available: https://doi.org/10.2307/25148625

[19] K. Peffers, T. Tuunanen, M. Rothenberger and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems,* vol. 24, no. 3, pp. 45-78, 2007. Available: https://doi.org/10.2753/mis0742-1222240302

[20] I. Bider, P. Johannesson and E. Perjons, "Design science research as movement between individual and generic situation-problem-solution spaces," in *Organizational Systems. An Interdisciplinary Discourse,* Springer, 2013, pp. 35-61. Available: https://doi.org/10.1007/978-3-642-33371-2_3

[21] J. Anderson, B. Donnellan and A. Hevner, "Exploring the Relationship between Design Science Research and Innovation: A Case Study of Innovation at Chevron," in *Communications in Computer and Information Science, 286,* 2012. Available: https://doi.org/10.1007/978-3-642-33681-2_10

[22] M. Sein, O. Henfridsson, S. Purao, M. Rossi and R. Lindgren, "Action Design Research," *MIS Quarterly*, vol. 35, no. 1, p. 37–56, 2011. Available: https://doi.org/10.2307/23043488

[23] I. Bider, A. Karapantelakis and N. Khadka, "Building a High-Level Process Model for Soliciting Requirements on Software Tools to Support Software Development: Experience Report," in *Short Paper Proceedings of the 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM 2013). CEUR, Vol. 1023*, Riga, Latvia, 2013.

[24] W. W. Royce, "Managing the Development of Large Software Systems," in *Proceedings of IEEE WESCON*, 1970.

[25] M. Polanyi, Knowing and Being, Chicago: University of Chicago, 1969.
Available: https://doi.org/10.1126/science.168.3938.1440

[26] I. Nonaka, " A dynamic theory of organizational knowledge creation," *Organ. Sci.*, vol. 5, no. 1, p. 14–37, 1994. Available: https://doi.org/10.1287/orsc.5.1.14

[27] C. de Souza, S. Quirk, E. Trainer and D. Redmiles, "Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies," in *Proceeding of the 2007 international ACM conference GROUP'07*, 2007. Available: https://doi.org/10.1145/1316624.1316646

[28] M.-A. Storey, Č. D and D. German, " On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework," in *Proceedings of the 2005 ACM symposium on Software visualization*, 2005. Available: https://doi.org/10.1145/1056018.1056045

[29] M. Stone, 2006. [Online]. Available: http://www.perceptualedge.com/articles/b-eye/choosing_colors.pdf. [Accessed 06.07.2018.]