Functional Decomposition of a Socio-Technical System: What is Missing?

Ilia Bider

Department of Computer and Systems Sciences (DSV), Stockholm University, Stockholm, Sweden

Ilia@dsv.su.se

Abstract. The paper discusses the needs of new expressive means for modeling a socio-technical system as consisting of functional components that are connected to each-other through the output-input relationships. The discussion is based on an example of depicting so-called feedback connections between the functional components of a system. The need to introduce a feedback connection arises when two connected components are heavily dependent on the intellectual activity of people who man the components. In such a situation, there is a risk that the output from one component could be misinterpreted by the component which takes it as an input. Using a simplified example of a software development project, the paper introduces the notion of a feedback connection and discusses the ways it could be realized in a socio-technical system.

1 Introduction

A socio-technical system differs from other types of systems in that it has human participants performing essential tasks inside the system. However, as any other system, a socio-technical system can be modeled as a consisting of a number of components interacting with each other, and the systems' environment t. In this paper, we do not consider the type of decomposition that differentiates social and technical components of the system [1]. Instead, we look at functional components of socio-technical system, which can be thought of as different teams of people completing different tasks that contribute to proper functioning of the system as a whole.

In a model with a functional decomposition, each component can be defined as having inputs and outputs, while connections between the components are modeled as outputs of some components serving as inputs for other components. There are several notations that can be used for depicting a system that consists of functional components connected via output-input relationships. These notations can be used for functional decomposition of purely technical systems, as well as the systems that include human activity. The most typical notation of this sort is IDEF0 [2] used for functional modeling of an enterprise/organization. To the same class belong workflow-based notations, e.g. BPMN [3], though they are less suitable for explicit representation of

output-input connections, as they are more focused on ordering the functional components in time sequence.

The question arises, whether the existing notations are sufficient for functional decomposition of socio-technical systems. In this paper, we will discuss one feature that is normally missing in the existing notations – representation of feedback connections. This is to demonstrate that notations/modeling techniques with better expressive power are needed for functional decomposition of socio-technical systems.

A functional model that depicts output-input relationships between the components can be useful for various purposes, for example, for examining logistics, including the informational one, between the components. However, such model has limitations as it presumes that an output produced by one component can be unambiguously interpreted by a component for which it serves as an input. This can be true in some cases, e.g., when an output of a technological department is a program controlling the automated equipment of the production line. However, when (a) both the outputting component and the component which accept the output rely on human actions, and (b) the output cannot be totally formalized to exclude possible ambiguities. Case (b) often happens when an output is a result of intellectual work that only partly can be strictly specified. A software engineering project is a typical example where such ambiguities are norm rather than exception.

In case when a risk for misinterpretation between the humanly manned components exists, there is a need to introduce an additional channel of communication between the components to mitigate the risk. This channel, which we call a feedback channel, needs to be explicitly represented in the model so that the presence and the quality of it can be properly analyzed. Not having feedback channels in the functional model used for analysis of existing problems, or planning organizational change may lead to a disaster. In the first case, a cause of the problem can be missed, and wrong action can be planned, like trying to more formalize the output, instead of introducing or improving a feedback channel. In the second case, organizational change can be planned so that the existing feedback channels become broken, while the new ones are not provided.

In this position paper, using a simplified example, we introduce the concept of feedback connection between functional components of a socio-technical system. The introduction is done in a way to be understandable for non-specialists in socio-technical systems or functional/enterprise modeling. We believe that our explanation of feedback can be used for raising awareness of the need to represent feedback in a functional model of a socio-technical system. After introducing the notion of feedback connection between functional components of a socio-technical system, we discuss the ways the feedback can be implemented: either via proper technical infrastructure or proper social structure of the system or both.

The material presented in this paper is based on the author's engagement in developing a modeling technique, called step-relationship model, for building high-level business process models [4,5]. The notion of feedback presented in this paper first appeared in [4] under the name weak dependency. It was later further developed when applying the step-relationship model to modeling two distributed software develop-

ment projects, see [6,7]. The term feedback connection was introduced in [7], along with its description that was adapted for this paper and presented in the next section.

2 Introducing the concept of feedback connection in socio-technical systems

To introduce the concept of feedback in socio-technical systems, we use an example of software development project in its simplified form presented in the diagram of Fig. 1. 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*[1]. Behind each component, there is a team of professionals completing the task of the corresponding component using specific methods, e.g. for design or coding, and tools, e.g., compilers, version management systems, etc. The teams that belong to different component may or may not intersect.

The functional components in Fig. 1 are related to each other through output-input relationships that are represented in Fig. 1 as arrows going from one component to another.

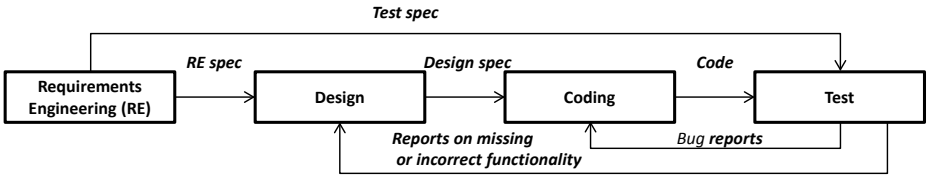

Fig. 1. Simple model of a software project as a system (note that connections between the project and its environment are not shown); adapted from [7].

The diagram in Fig. 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 delivered as its input. Alternatively, the components can work in parallel delivering its output in smaller portions as inputs to the next component in line. Also, the outputs could be arranged and delivered in various ways. For example, 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 Fig. 1 can work satisfactorily if the inputs received can be interpreted unambiguously by the receiving components. This is difficult, if ever possible, 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)" [8]. To minimize the risk that the input is interpreted in a different way than it was meant, a negative (corrective) feedback needs to be introduced into the system. The work of such feedback is presented in Fig. 2.

[1] Description of a more complex and real example, can be found in [6,7]

Fig. 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 [9]; 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 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 equal to *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 Fig. 2, the controller compares *Understanding A* with *Understanding B* and adds additional correcting input to *B*.

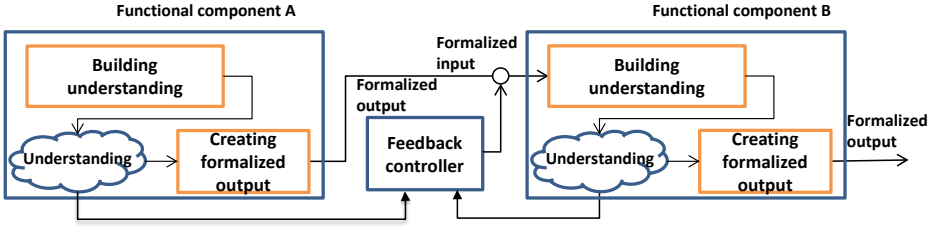

Fig. 2. Negative feedback between two components of the system (adapted from [7]).

The work of Feedback controller is illustrated in Fig. 3. The axes in Fig. 3 represent the space of all possible interpretations of Formalized input *B*. This space is shown 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 Fig. 3 as it develops, starting as a wider circle in the 1st iteration that in the end should be narrowed to a point, hopefully coinciding[2] 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 Fig. 3, there is no need for the *Feedback controller* to take an action. However, as soon as the coverage vanishes, as in the 2nd iteration in Fig. 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 Fig. 3.

The feedback connection between the system's components is needed as a complement for any output-input relationship that cannot be understood unambiguously. We represent this connection with a dashed double arrow connecting the functional

[2] Understanding *A* and understanding *B* needs to be equal only in the dimensions that are relevant for both component *A* and component *B*. Each of these components may have additional dimensions to its *understanding* that are not important for the other component. For example, *Design* has dimensions related to the design of the system, which is normally considered of no importance for *RE* (separation of the problem and solution domain).

blocks with a label of what type of information is to be used by the feedback controller. For example, adding feedback connections to the system diagram on Fig. 1 produces the system diagram of Fig. 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.

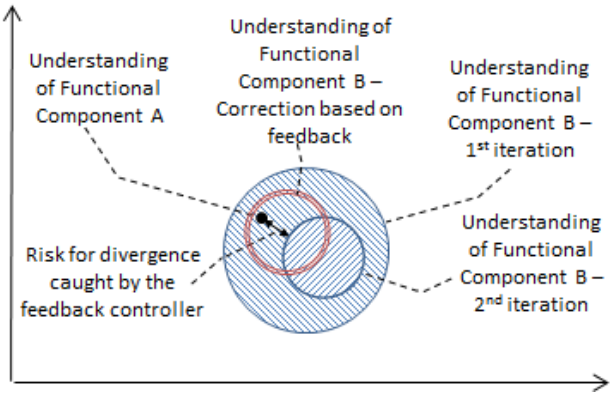

Fig. 3. Interpretation space and the work of the feedback controller (adapted from [7]).

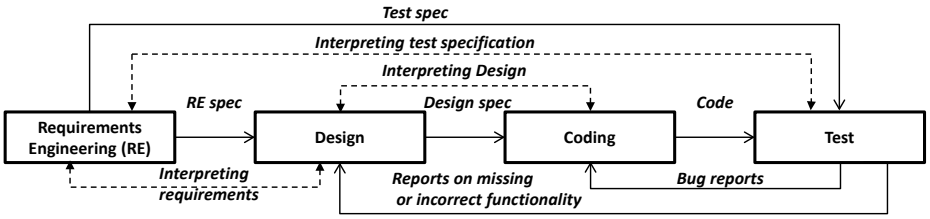

Fig. 4. Diagram from Fig. 1 complemented with feedback connections (adapted from [7]).

3 Implementing feedback connections in a socio-technical system

Naturally, in a socio-technical system, 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 Nanako's theory of knowledge creation [10].
- 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 that discovery of divergence, and thus the need for correction, falls in the area of responsibility of team *B*. The discovery can, for example, happen when (a) team *B* starts having doubts that they understand the input properly, (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. In case (a), the behavior of team *B* is somewhat reactive, they have already interpreted their formalized input, but suspect that that the divergence have happened and need a corrective signal. In case (b), the behavior of team *B* is somewhat proactive, as its initiates receiving a corrective signal before actual divergence happens.

Note, also, that during the feedback communication initiated by team *B*, team *A* can come to a conclusion that their *understanding* may be incorrect/incomplete, which may lead to initiating a feedback communication channel with a component from which they have got their own input. Thus, initiation of the feedback communication between two components can result in a recursive wave of initiations of feedback channels backwards in order for the first feedback channel to produce a proper corrective signal.

4 Concluding remarks: new modeling techniques are needed for functional decomposition of socio-technical systems

In Sections 2, we have introduced and explained the notion of feedback connection in socio-technical systems, which, hopefully, could bring attention of business analysts and organizational change planners to the issues related to the lack or insufficient quality of these connections. Without feedback connections, a socio-technical system may rely too heavily on the formalized outputs as inputs for the next functional components, and therefore may not provide means for correcting misinterpretation. For example, suppose an *RE* team is dissolved after completing its work, or moved to another project and becomes unavailable. Such a situation may leave the divergence of *Understanding A* and *B* in Fig. 3 uncorrected, which can result in erroneous software being produced by the project.

As was discussed in Section 3, there are several ways for arranging feedback connections in a socio-technical system, e.g.: (1) through the proper arrangement of the social structure of the system, e.g. through intersecting teams, and (2) through its technical infrastructure, e.g., providing a groupware system that stores all intermediate results produced by each team and makes them available for other teams, or (3) a combination of 1 & 2. These means are completely different from those used to arranging feedback in the pure technical systems.

Feedback connections is only one type of phenomena that need to be represented when modeling a socio-technical system as a decomposition of functional components. Some other issues, related to Global Software Development (GSD), are listed in [7], e.g. heterogeneousness of components teams. The particularities of socio-technical systems make the existing modeling languages and notations for functional decomposition insufficient for modeling this type of systems. Either the existing languages and notations needs to be extended or the new ones are to be introduced. The modeling technique we introduced in [7] for GSD can serve as an example of such techniques. However, whether this technique is universal enough to be suitable for other types of socio-technical systems remains an open question.

Acknowledgements: Many thanks to all people participated in the projects that lead to the discovering and, partially, solving the problems discussed in this paper, especially to E. Perjons, A. Karapantelakis, B. Rutkowska, H. Otto. The author is also grateful to the anonymous reviewers whose comment helped to improve the text.

References

1. Mumford, M.: The story of socio-technical design: reflections on its successes, failures and potential. Information Systems Journal 16(4), 317–342 (2006)
2. NIST: Integration definition for function modeling (IDEF0), Draft Federal Information Processing Standards, Publication 183, 1993. (Accessed February 2015) Available at: www.idef.com/downloads/pdf/idef0.pdf
3. OMG: Business Process Model and Notation (BPMN), Version 2.0.2, Object Management Group (OMG), Document formal/2013-12-09, December 2013. (Accessed February 2015) Available at: http://www.omg.org/spec/BPMN/2.0.2/PDF
4. Bider, I., Perjons, E.: Preparing for the Era of Cloud Computing: Towards a Framework for Selecting Business Process Support Services. In : Enterprise, Business-Process and Information Systems Modeling, LNBIP, Vol 113, Springer, pp.16-30 (2012)
5. Bider, I., Perjons, E.: Design science in action: developing a modeling technique for eliciting requirements on business process management (BPM) tools. Software & Systems Modeling, http://link.springer.com/article/10.1007/s10270-014-0412-6 (2014)
6. Bider, I., Karapantelakis, A., Khadka, N.: 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, pp.70-82 (2013)
7. Bider, I., Otto, H.: 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 (ICGSE), forthcoming, Ciudad Real, Spain (2015)
8. Jacobs, D.: Requirements Engineering so Things Don't Get Ugly. In : ICSE 2007 Companion, Minneapolis, MN, US, pp.159-160 (2007)
9. Polanyi, M. S.: Knowing and Being. University of Chicago, Chicago (1969)
10. Nonaka, I.: A dynamic theory of organizational knowledge creation. Organ. Sci. 5(1), 14–37 (1994)