The Journal of Systems and Software 109 (2015) 229–242

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A general theory of software engineering: Balancing human, social and organizational capitals

Claes Wohlin [a,*], Darja Šmite [a], Nils Brede Moe [a,b]

[a] Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
[b] SINTEF, NO-7465 Trondheim, Norway

ARTICLE INFO

Article history:
Received 4 December 2014
Revised 30 July 2015
Accepted 7 August 2015
Available online 13 August 2015

Keywords:
Software engineering theory
Intellectual capital
Empirical

ABSTRACT

There exists no generally accepted theory in software engineering, and at the same time a scientific discipline needs theories. Some laws, hypotheses and conjectures exist, but yet no generally accepted theory. Several researchers and initiatives emphasize the need for theory in the discipline. The objective of this paper is to formulate a theory of software engineering. The theory is generated from empirical observations of industry practice, including several case studies and many years of experience in working closely between academia and industry. The theory captures the balancing of three different intellectual capitals: human, social and organizational capitals, respectively. The theory is formulated using a method for building theories in software engineering. It results in a theory where the relationships between the three different intellectual capitals are explored and explained. The theory is illustrated based on an industrial case study, where it is shown how decisions made in industry practice are explainable with the formulated theory, and the consequences of the decisions are made explicit. Based on the positive results, it is concluded that the theory may have a good explanatory power, although more evaluations are needed.

1. Introduction

Software development is a very knowledge-intensive activity. It is an engineering endeavour involving a lot of design, and the production is relatively simple. To develop software many different people interact within an organization. Thus, software development is hugely dependent on people (DeMarco and Lister, 2013). However, people alone are insufficient. Software development is to a very large extent a team effort, and hence the interaction between people and the complementarity in expertise are prerequisites to be successful. Furthermore, the organization in which the people work provides the infrastructure and environment to be able to leverage on the individual skills and their combined value. The organizational aspects relate to processes, methods, techniques and tools being part of the work environment. These three aspects are captured in the concept of intellectual capital. The objective of the paper is to formulate a general theory of software engineering from empirical observations of how industry actively works with human, social and organizational capitals (components of intellectual capital) to help explaining and reasoning about combinations of intellectual capital components (ICCs) to be successful in software development.

Intellectual capital may be defined as: "the sum of all knowledge firms utilize for competitive advantage" (Nahapiet and Ghoshal, 1998; Youndt et al., 2004). The sum of all knowledge means that the concept of intellectual capital encompasses all assets available to a company. Different divisions of intellectual capital into components exist. Here it is chosen to use the division discussed by Youndt et al. (2004). Some alternative divisions are briefly introduced in Section 2.1. Youndt et al. (2004) divide the general concept of intellectual capital into three ICCs: human capital, social capital and organizational capital. They are depicted in Fig. 1 together with the main level where it primarily resides, i.e., individual, unit and organizational, respectively. The ICCs are described in Section 2.

Here, the concept of a unit is used to denote an entity utilizing the three components of intellectual capital: human, social and organizational capitals, respectively. The unit may be a team, a department or any other entity for which it is relevant to discuss the concept of ICCs. A unit includes people, who possess a certain level of human capital through their experiences and expertise. It also has a social capital both in terms of how it can leverage on the social interaction within the unit, and how it uses its external contacts to create value. The external contacts and networks may include customers, internal

* Corresponding author. Tel.: +46 455 385820.
E-mail addresses: claes.wohlin@bth.se (C. Wohlin), darja.smite@bth.se (D. Šmite), nils.b.moe@sintef.no (N.B. Moe).

http://dx.doi.org/10.1016/j.jss.2015.08.009

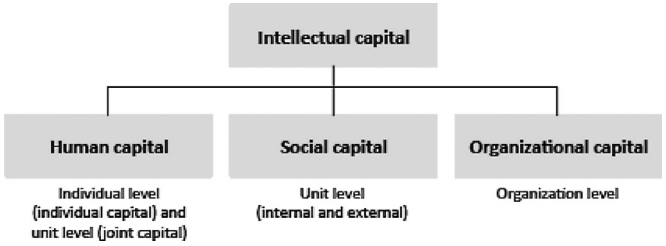

Fig. 1. Intellectual capital and its three components.

people in the organization, or external networks (including communities of practice, blogs and other external contacts and information). The unit exists in a context, which provides the organizational capital, for example, the support available to software engineers in terms of infrastructures. The latter includes all aspects of an organization that remain if removing all humans.

From the above reasoning, it becomes clear that the different components of intellectual capital are what make it possible to develop software. Based on this observation, this article contributes with formulating a theory of software development that captures the balancing of the ICCs that software organizations use in practice. Thus, the formulation of the theory is based on observations of practice and the insight that although organizations are different, they have a similar challenge. They need to balance the ICCs to be able to conduct their business in a cost-effective and competitive way. Balance refers to compensating loss in one ICC with improving either the same ICC or at least one of the other ICCs. The article presents the theory formulated and its constituents. Furthermore, it illustrates the theory in a real industrial case and also provides some examples taken from industrial collaboration.

The remainder of the article is structured as follows. Related work is presented in Section 2. Section 3 introduces the theory based on the steps recommended by Sjøberg et al. (2008). The theory is exemplified and illustrated by an empirical case in Section 4. In Section 5, a discussion is provided and the article is concluded in Section 6.

2. Related work

2.1. Intellectual capital and software engineering

In software engineering, there has been much discussion about how to manage knowledge, or foster "learning software organizations". In this context, Feldmann and Althoff have defined a "learning software organization" as an organization that is able to "create a culture that promotes continuous learning and fosters the exchange of experience" (Feldmann and Althoff, 2001). Dybå places more emphasis on action in his definition: "A software organization that promotes improved actions through better knowledge and understanding" (Dybå, 2001).

Because software development is knowledge-intensive work, intellectual capital is a particularly relevant perspective for software companies. Intellectual capital is called the main asset of software companies (Gongla and Rizzuto, 2001; Rus and Lindvall, 2002). It is seen as a construct with various levels (individual, network, and organizational) (Youndt et al., 2004). As mentioned above, Youndt et al. (2004) divide intellectual capital into three components: human, social and organizational capitals. This is not the only proposal for how to describe intellectual capital. Stewart (2001) describes the essential elements or assets that contribute to the development of intellectual capital as:

- Structural capital: Codified knowledge that can be transferred (e.g., patents, processes, databases, and networks).
- Human capital: The capability of individuals to provide solutions (e.g., skills and knowledge).
- Customer capital: The value of an organization's relationships with the people with whom it does business and share knowledge with (e.g., relationships with customers and suppliers).

The possession of each of these assets alone is not enough. Intellectual capital can only be generated by the interplay between them. Therefore, Willcocks et al. (2004) propose a framework, which also includes a fourth kind of ICC—social capital. Social capital helps to bring structural, human and customer capital together and encourages interplay among them.

Here it has been chosen to use the division of intellectual capital advocated by Youndt et al. (2004) for two main reasons. First, we agree with Youndt et al. that organizational capital is more fitting than the term structural capital because this is capital the organization actually owns (human capital can only be borrowed or rented). Second, both frameworks define social capital to consist of knowledge resources embedded within, available through, and derived from a network of relationships. We support Youndt et al.'s argument that such relationships are not limited to internal knowledge exchanges among employees, but also extend to linkages with customers, suppliers, alliance partners, and the like. We then see customer capital as part of social capital.

Creating intellectual capital is more complicated than simply hiring bright people. The importance of intellectual capital can be demonstrated by the ratio of intellectual capital to physical capital involved in the production of software. Symptomatically, the ratio of the software development industry is found to be seven times the ratio of other industries that are heavily reliant on physical capital, such as the steel industry (Bontis, 1997, 1998; Tobin, 1969). In a study on intellectual capital in Systematic Software Engineering Ltd, Mouritsen et al. (2001) found that the main motivation for understanding the different elements of intellectual capital was to make the company's knowledge resources and key competency areas visible and to monitor management's efforts to develop these. Also, management wanted to establish a new basis for deciding about the future of the company.

Youndt et al. (2004), through their review of intellectual capital, conceptualize intellectual capital through the three distinct components: human, social, and organizational. Human capital refers to individual employee's knowledge, skills, and abilities. In software engineering these are often associated with technical skills including design expertise, domain knowledge and product knowledge (Faraj and Sproull, 2000; Moe et al., 2014). Organizational capital represents institutionalized knowledge and codified experience stored in databases, routines, patents, manuals, infrastructures, and the like. Many traditional software companies that follow plan-driven approaches believe that a good process leads to a good product, and thus standardized and well-documented processes support developers, while interaction among software developers is usually minimized. Finally social capital consists of knowledge resources embedded within, available through, and derived from a network of relationships possessed by an individual or a social unit. Social capital is both the network and the assets that may be mobilized through that network (Bourdieu, 1986). It enables achievements that would be impossible without it or could only be achieved at an extra cost. Also, because social capital increases the efficiency of information diffusion, a company can have less redundancy in, e.g., skills or roles if the social capital is strong. An organization supports the creation of social capital when it brings its members together in order to undertake their primary task, to supervise activities, and to coordinate work, particularly in the context requiring mutual adjustment.

Different ICCs belong on different levels—individual, unit or organizational levels. While human and organizational capital components are rather straightforward, social capital is a more complex phenomenon. In the research on social capital, scholars have tended to adopt either an external viewpoint (the relations an actor

Table 1
Types of intellectual capital based on the synthesis by Youndt et al. (2004) and examples by Moe et al. (2014).

Intellectual capital	Definition	Specific examples
Human capital	The "skill, knowledge and similar attributes that affect particular human capabilities to do productive work" which can be improved through health facilities, on-the-job training, formal education and study programmes (Schultz, 1961, pp. 8–9). This capital resides with, and is utilized by individuals.	Domain knowledge; Knowledge about programming, practices, languages and architecture.
Social capital	The actual and potential resources embedded within, available through, and derived from the network of relationships possessed by an individual or social unit. According to Nahapiet and Ghoshal (1998) social capital have three main dimensions: structural (including network ties, network configuration and appropriable organization), cognitive (including shared codes and language and shared narrative) and relational (including trust, norms, obligations and identification) (ibid).	Relationship between team-members, network of experts, participating in external forums, communication coding and architectural conventions; Trust in people outside the unit; Pride of and identification with product.
Organizational capital	The possessions remaining in the organization when people go home after work. This includes the "institutionalized knowledge and codified experience residing within and utilized through databases, patents, manuals, structures, systems and processes" (Youndt et al., 2004).	Software source code; Documentation; Documented work processes.

maintains with other actors) or an internal viewpoint (the structure of relations among actors within a grouping) (Adler and Kwon, 2002). The distinction between the external and internal views on social capital is, to a large extent, a matter of perspective and unit of analysis. The relations between an employee and colleagues within a unit are external to the employee but internal to the unit. Because, the capacity for effective software development in a unit is typically a function of both its internal linkage and its external linkage to other units and experts, we have adopted the view of Nahapiet and Ghoshal (1998), who describe the social capital as both internal and external to a unit.

A summary of the definitions of the three different components of intellectual capital as described by Youndt et al. is given in Table 1. In this table, the information on how the concepts synthesized by Youndt et al. (2004) link to software engineering is provided as based on Moe et al. (2014).

It is possible that organizations can develop these individual dimensions of intellectual capital independently. For example, targeting hiring strategies of experts in specialized areas could help to acquire human capital. Similarly, procuring particular databases or investing in the installation of specific systems and processes could create organizational capital. Accumulation of social capital can be fostered by, e.g., establishing communities of practice and regular forums for interaction. However, there are strong interdependencies in the creation, development, and leveraging of the three components of intellectual capital. Organizational learning theorists (Nonaka and Takeuchi, 1995; Schön, 1983) point out that organizations do not create knowledge; rather people, or human capital, is the origin of all knowledge. And when people share or exchange tacit knowledge, this is most likely to be done through discussions. Also it is suggested that: "individual learning is a necessary but insufficient condition for organizational learning" (Argyris and Schön, 1996). In order for organizational level learning to occur, individuals should exchange and diffuse shared insights and knowledge, that is, use their social capital. Also, social capital helps in creating new knowledge among individuals and for organizational learning to occur. Therefore social capital has been found to be important in the development of human capital. And ultimately, much of the knowledge individuals create through human capital and diffuse through social capital becomes codified and institutionalized in organizational databases, routines, systems, manuals, and the like, thereby turning into organizational capital.

2.2. Theories in software engineering

The need for a firm theoretical basis for software engineering has been emphasized since the infancy of the area as exemplified by Freeman et al. (1976). Specific theories for software engineering were also proposed such as Musa's (1975) theory with respect to estimation of software reliability. The field has progressed since but there are still no commonly accepted theories for software engineering (Ralph et al., 2013). The need to build theories has been emphasized by, for example, Sjøberg et al. (2008) and more lately by Johnson et al. (2012). Thus, there is a drive to obtain a stronger theoretical foundation in software engineering.

In addition to theories, laws and empirical observations have helped to increase the understanding of the discipline as described by Endres and Rombach (2003). Some examples include Conway's law (Conway, 1968) with respect to the relationship between system structure and organization, and Lehman's laws (Lehman, 1979) on software evolution.

Conway's law describes how the organization and software structure mirror each other. Endres and Rombach (2003) take it one-step further and explain how the law can be interpreted as a theory, since there is a logical explanation to the law. Their explanation is that software system development is more of a communication problem than a technical problem, and hence the organization and the software structure are highly likely to be aligned.

Lehman puts forward five laws on software evolution (Lehman, 1979). His first two laws are used as examples here. The first law states that a system that is used will be changed. The second law relates to complexity and describes how a software system will become more complex as it evolves if specific actions are not taken to reduce complexity. Both these laws have logical explanations, and hence Endres and Rombach describe how they can be interpreted as theories.

Some of the laws described by Endres and Rombach (2003) are well established in both research and practice and others are not, and some of them can be turned into theories. However, since software development is a very knowledge-intensive activity involving a lot of people, there is a need for a theory that relates software engineers, software engineering team(s), software engineering project(s), or software engineering organization(s) etc., to the development and evolution of software system(s). A software engineering theory taking human, social and organization capitals into account in software engineering is lacking. This article attempts to fill this gap by contributing with a theory taking a broad perspective on software engineering, including human, social and organizational capitals.

3. Theory formulation

3.1. Background

The theory is inspired by the authors' observations of industry practice. Research conducted by the authors in the past five years has

brought about vivid manifestations of the ways software organizations approach ICCs in practice (Moe et al., 2014; Šmite and Wohlin, 2011, 2012). Specifically, the research conducted has focused on how several industrial partners practice global software development, execute software product transfers (relocation of development from one team or set of teams to a new team(s), often in different locations) and manage the challenges related to such transfers (ibid). As a side effect, it has over the years been observed how companies make decisions to compensate for issues related to consequences of transfers, often in terms of the ICCs. In general, it has been observed that if actions are not taken, a transfer will mean a loss of experience and expertise in relation to the product, and hence a decline in human capital (in this case product knowledge and potentially domain knowledge), which has often a direct impact on development capabilities and a secondary impact on quality. Furthermore, it has been observed that after a transfer the new teams involved with a product are more dependent on the documentation and support in the organization than the experienced developers used to be before the transfer, i.e. the new teams depend more heavily on the organizational capital and the social capital (in particular in relation to the teams conducting the development before the transfer). Some specific examples:

- Example 1: The product documentation was deemed insufficient for a transfer, and hence nine person-months were spent on improving the product documentation before transferring a software product (Šmite and Wohlin, 2010).
- Example 2: A gradual transfer (Wohlin and Šmite, 2012) was conducted, i.e., joint development between sites was organized before transferring the software product. This resulted in a competence build up in the receiving site, while leveraging on the presence and active involvement of the original developers.
- Example 3: Temporal relocation of experts from the sending site with the product to the receiving site has been seen as a common practice to ensure the presence and accessibility of expertise and to transfer knowledge to the teams receiving the software product (Šmite and Wohlin, 2010).

The three examples together with access to both product and project artifacts, and continuous discussions with practitioners in different roles at the companies have resulted in a general observation: companies try to compensate a potential loss in one component of intellectual capital with different countermeasures, either in relation to the same component of intellectual capital (e.g., human capital—send an expert) or in another component of intellectual capital (e.g., organizational capital—improve the software product documentation, or social—foster interaction with remote experts from the original site). Thus, it has been observed that there is an interplay between different components of intellectual capital that companies try to master to ensure that the setting for the software product development or evolution is fit for its purpose, including the type of tasks to accomplish and the objectives in terms of, for example, delivery time and quality.

Based on the observations from the long-term collaboration with industry, in particular in the area of global software engineering, the objective here is to formulate a general theory for software engineering including the different components of intellectual capital.

3.2. General theory formulation

Based on the above, the following theory is put forward, the theory of:

Balancing Human, Social and Organizational Capitals for Software Development and Evolution

Software may be developed and evolved by having different combinations of the components of intellectual capital, i.e., a combination of human, social and organizational capitals. Many different combinations of the capitals may help to solve a given task with a specific objective in a given context. Changes in the task, objective or context may result in the changes in demand of the intellectual capital, or changes in one or two of the components of intellectual capital may force a need to change one or two of the other components, to adjust to the new situation. A balancing of the different components of intellectual capital is needed to ensure that software engineers, teams, or organizations are sufficiently equipped to carry out the task, with a specific objective at hand, in the given context.

Companies strive for finding the right balance, which in a cost-efficient way gives a sufficient level of intellectual capital to carry out the tasks under the given constraints (features, time, cost and quality) with a specific objective in the given context. Too low intellectual capital means that the tasks cannot be carried out adequately, and too much intellectual capital results most likely in the costs being higher than desired. This gives a delicate balance to master for companies developing software.

According to the different types of theories described in Sjøberg et al. (2008), the theory of balancing the ICCs for software development and evolution is primarily explanatory, although it may also help managers to answer "what if"—questions, and hence at least partially help in prediction, or at least in reasoning about the effects of changes. The theory is formulated based on abduction from observations in industry with the objective to capture and help explain the observations. The theory is presented below according to the following steps:

1. Constructs of the theory.
2. Propositions of the theory.
3. Explanations to justify the theory.
4. Scope of the theory.
5. Testing the theory through empirical research.

Sjøberg et al. (2008) proposed these steps as suitable for formulating theories (in software engineering). Steps 1–4 are presented in Sections 3.3– 3.6, followed by a summary of the theory and a discussion about its use in practice. An empirical case study is presented in Section 4 to illustrate the theory, and hence act as a starting point for step 5 above.

3.3. Constructs

The constructs of the theory relate to the building blocks that make up the theory. Thus, the question is: What are the basic elements?

When developing software, it is possible to have different levels of ambition from an organizational perspective (Rajlich and Bennett, 2000); ambition is here used in a general sense. For example, if having an old piece of software that is intended to be phased out shortly and replaced with a new software system, then the ambition of the organization may not be very high. It may be sufficient to keep it afloat and do some corrective maintenance, and it may not be perceived as critical to fix any issues immediately. Thus, the organization has a quite low ambition level. Another example may be when launching a new software system and trying to increase the market share for a specific type of product. In this case, it may be very important to have a high quality product and if problems occur then they should be addressed very quickly. Thus, the ambition level of the organization may be considerably higher than in the first case. This leads to a construct denoted as *objective*, which relates to the ambition level in terms of performance levels; see Section 3.3.1, where objective is focused on a specific performance level. This leads to *performance* being the second construct. Meeting the objective is referred to as success, where success in this context refers to the ability to conduct a software development task under a given objective with the intellectual capital available meeting the goals set by the organization. Thus, it

is chosen to use "success" in a generic sense given that different organizations may have different criteria for being successful in their software development.

The actual development to be conducted is referred to as the *task*, which is the third construct for the theory. Some tasks are more challenging than others, and hence the objective should be set in relation to the task to conduct. For example, to work with corrective maintenance is a different task than adding new features to a software system. These tasks may be differently complex to carry out and hence the task to be conducted should be taken into account when deciding how to carry out the development. However, it should be noted that task complexity/difficulty is hard to measure objectively, and in particular the ability to conduct a task is highly dependent on the intellectual capital available. Thus, it is chosen to have task as a construct and not task complexity/difficulty, since the conduct of the task is handled through the ICCs in the theory, see Section 3.4. The task is connected to the objective through the development and evolution levels (performance levels) presented in Section 3.3.1.

To be able to conduct the task with the given objective, the intellectual capital should be carefully considered, in particular given that software development is a very knowledge-intensive discipline. For example, it may be obvious that taking a group of new graduates and letting them form a new team to develop a new feature for an existing large, complex and poorly documented software system may be an overwhelming task for them. This illustrates that a certain intellectual capital is needed to be able to perform the task. As described above, intellectual capital has been categorized into different components by different researchers. Here, it is chosen to follow the division by Youndt et al. (2004), where intellectual capital is divided into: *human, social and organization capitals*. These ICCs make up three important constructs of the theory. These three constructs are presented in Table 3 and discussed in more detail in Section 3.3.2.

Finally, the mixture of the objective and the task sets the target for the needed intellectual capital. In total, the intellectual capital has to be at a certain level to enable that the task may be performed in relation to the objective set, and if meeting the objective it should be viewed as a success. Thus, the *performance* is a construct in the theory, since it ties together *objective* and *task* with the three ICCs: *human, social* and *organizational capitals*.

3.3.1. Performance levels

To describe the objective, five performance levels have been formulated, although in practice the scale is continuous. Furthermore, the levels are qualitative although numbers are associated with the levels to ease the discussion about them and to help in ordering the levels. The continuity of the scale has emerged in discussions with practitioners where it became evident that although being on one performance level (e.g., level 3), they were closer to one of the neighbouring levels than the other. However, discrete qualitative levels were used as a starting point for the discussions with industry. The intention is to capture the ambition of an organization related to desired performance for a given task. An organization is expected to have different ambition levels for different software development projects or products, and it may also vary over time. In Table 2, five performance levels have been defined with five being the highest and most ambitious level, i.e., the organization tries to ensure the level by managing the intellectual capital accordingly. The levels relate to the capability to meet the objectives set by the organization developing the software.

3.3.2. Intellectual capital

As stated earlier the intellectual capital may be described as consisting of three components. The human capital captures the skills, knowledge, expertise and experience of the individuals and unit's capital. Social capital is concerned with the network outside and inside the unit (e.g., outside and inside the team). The third component is the organizational capital that is the assets in the organization without the people. This includes documentation in relation to the actual software being developed, but also supporting aspects such as processes, tools and culture. These three constructs are divided into areas and specific aspects in relation to each capital as exemplified in Table 3. It should be noted that ICCs are intended to cover all aspects of knowledge available to a company, i.e. it is the "sum of all knowledge" (Nahapiet and Ghoshal, 1998; Youndt et al., 2004). The sum should not be viewed in mathematical terms; instead it should be seen as a metaphor for balancing the qualitative judgment of the aspects making up the different ICCs.

Table 2
Performance levels.

Level	Description
1	It is almost impossible to handle the task, and it takes a long time. The development is more or less in survival mode.
2	It is hard time to handle the tasks. Major problems occur more often than not.
3	The task requires some effort. Occasionally, major problems may occur. In most cases, it works quite smoothly.
4	The task is handled without any major problems.
5	The task is very easy to handle.

The theory is centred around these three ICCs and the processes of balancing them for performance on a software development task under a given objective. In relation to this, several things may be noted:

- In any situation involving more than one software developer, all three components are important.
- The qualities of each of these components, its categories and aspects for a given unit form the unit's intellectual capital profile.
- For any non-trivial software development task, there is a minimum "sum" of the components, and none of the components adds zero value. Unfortunately, there is no mathematical way of adding ICCs together quantitatively so the sum should be interpreted as a perceived combination of the "values" of the components.
- There is a maximum sum of the components.
- In normal cases, there is a sum of the three ICCs that is perceived as sufficient for the objective set and the task at hand. This is referred to as the target level to reach when combining the ICCs, which can be achieved through different intellectual capital profiles.

3.4. Proposition

The constructs interact through that a software development organization may set a goal of what to achieve in terms of what software to be delivered (task) and how well and fast it should be done (objective). The ability to reach the goal, i.e. to develop and evolve a software product, and thus the resulting performance is a combination of the objective, task and the sum of ICCs. Thus, the proposition is that the performance is a result of the objective, the task and the sum of ICCs. Fig. 2 illustrates the theory. The task is in the centre, and it is going to be performed with a given objective (desired level of performance) using the intellectual capital available. The objective sets the expectations on the conduct of the task, i.e. in terms of scope, quality, time and cost. The ICCs taken together facilitate the conduct of the task in solving it with respect to the objective. The outcome is a performance, which should be compared with the objective of the organization and based on their view of being successful in software development. The reasoning could be compared with requirements coming into software development (objective) and the available resources to implement the requirements (available intellectual capital). Performance is the outcome in terms of the objectives set by the organization.

Thus, Fig. 2 illustrates how the goal in terms of objective and the task and its difficulty require a certain combination of ICCs to reach

Table 3
The three intellectual capital components (ICCs) and examples of their categories and aspects.

ICC	Categories	Examples of aspects
Human capital	Skills and knowledge	*Technical skills (programming and tools, patterns, basic computer science principles)*
		Domain knowledge (including understanding of solutions to domain problems)
		Software product knowledge (program properties, existing software architecture, concept location within the code)
		Knowledge about ways of working (coding conventions, development tools etc.)
	Creativity	*Development of new, innovative ideas*
Social capital	The unit skills of working together	*Solving problems together*
		Making decisions together
		Shifting workload
		Common goals
		Performance of the unit
		Sharing knowledge within the unit
		Give each other feedback
		Knowing what others are doing
		Learning from experience
	External relations	*Collaboration with other units*
		Collaboration with experts
		Collaboration with customers
		Collaboration with product owners and program managers
		Networking through communities of practice
Organizational capital	Software	*Software source code*
		Software architecture
	Documentation	*Documentation supporting understandability and maintainability of the software*
		Process documentation
	Organization's culture	*Stories, rituals that contain valuable ideas, ways of working*
	General infrastructure	*Development environment*
		Knowledge-based infrastructure

the target. This is further illustrated in Fig. 3. To the left in Fig. 3, the starting point is the desired level of performance. The scale is in reality continuous, but for reasons of approximation and simplicity, it has been chosen to use five performance levels as described in Section 3.3.1. Here, the levels are not shown, since the objective is to illustrate the relationships rather than to describe a real case. The performance levels are used in the discussion of the usage of the theory in Section 3.6 and in the actual case described in Section 4. Once the performance level is set, the difficulty of the task has to be judged, and depending on the difficulty and the objective a certain sum of intellectual capitals is needed as shown with the arrow going from performance to intellectual capital. The needed intellectual capital becomes the target to be able to perform the task. If exactly meeting the target intellectual capital by balancing human, social and organizational capitals respectively, then the actual performance for the task at hand is equal to the objective. Fig. 3 illustrates the principal relationships, and some scenarios are discussed below to further illustrate the theory about the balancing of the different intellectual capitals to achieve the intended performance (objective for the task).

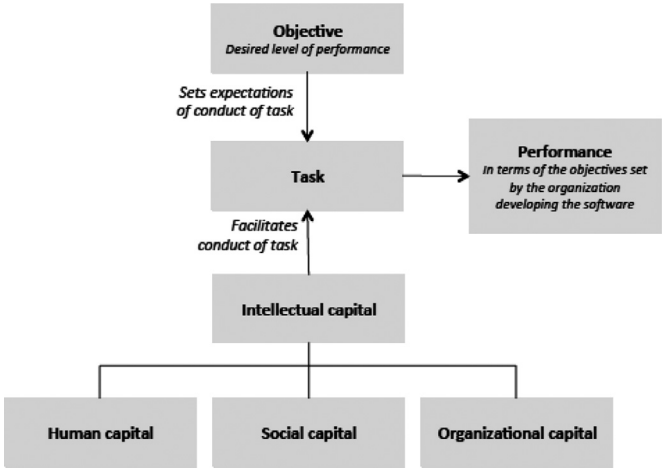

Fig. 2. Illustration of the theory proposition.

It is worth noting that the angle of the arrow going from performance to intellectual capital, and the arrow going back from intellectual capital to performance, will have the same angle, although not necessarily being in the same place as illustrated in Fig. 4 below. The actual angle of the arrows between performance and intellectual capital is given by the difficulty of the task. It should be noted that the difficulty of the task does not imply whether the arrow should go up, down or be on the same level. An easier task will of course have a lower requirement on the intellectual capital than a more challenging task. However, the arrow from performance can still go down for a challenging task, since the starting point for the arrow depends on the objective (in terms of performance level) and not the task as such.

Fig. 3 illustrates how it is possible to set an objective in relation to the performance levels introduced in Section 3.3.1. Given the objective and then taking the task into account, a certain target level of the intellectual capital is set. The target level indicates the level of intellectual capital needed to perform the task under the given objective. Given that the intellectual capital may be described in terms of three components: human, social and organizational capitals, the challenge is then to identify a combination of the ICCs or a suitable ICC profile that in total gives the intellectual capital needed.

Different scenarios may occur as illustrated in Fig. 4. If having too little intellectual capital (Fig. 4(a)), then the development and evolution will be challenged in relation to either fulfilling the task or reaching the objective in terms of performance levels. Fig. 4(a) illustrates this situation where the objective and task taken together point to a targeted intellectual capital (combination of human, social and organization capitals), which is higher than actually having. This is shown by the intellectual capital being lower than the target (illustrated with the arrow denoted "acquired sum of ICCs", and hence the performance will not be in accordance with the objective for the task to be conducted.

On the other hand, if having too much intellectual capital (see Fig. 4(c)), the development and evolution may go easier than required, which may be good, but it may result in being a too costly solution. For example, it may be too costly in terms of having too

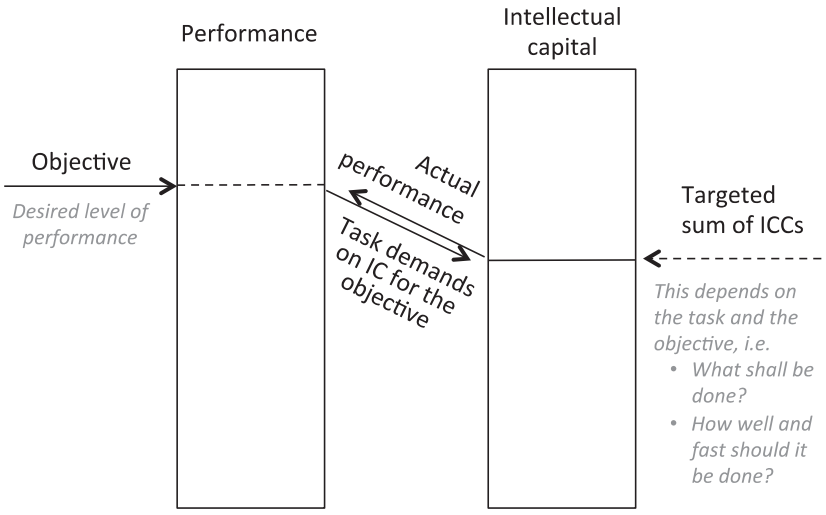

Fig. 3. A summary of the theory constructs and the proposition of the interaction.

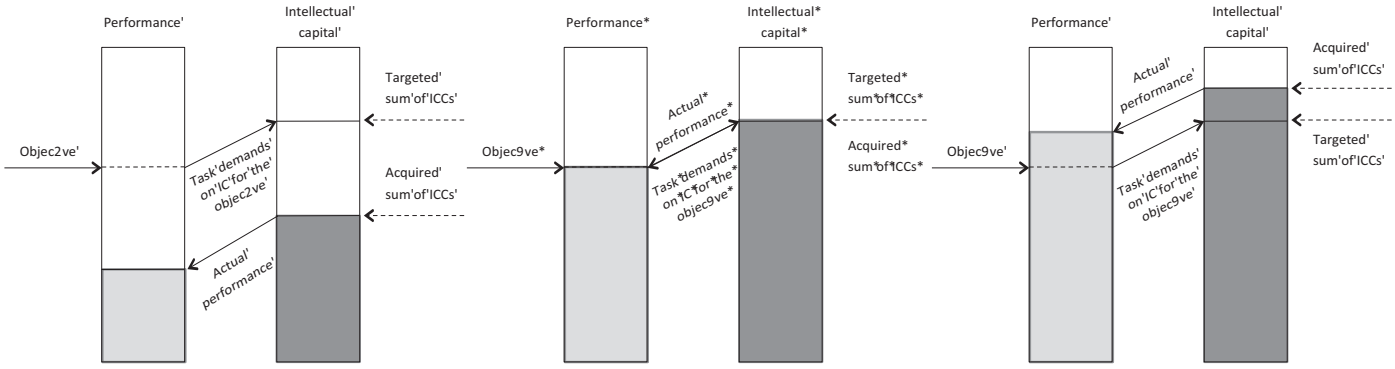

Fig. 4. (a) Targeted ICCs sum not reached. (b) Targeted ICCs sum reached. (c) Targeted ICCs sum is overreached.

many highly qualified people with long experience in the unit (human capital higher than the needs), investing too much in evolving or maintaining the social network (social capital higher than the needs), or putting too much effort into documenting well-documented software or refactoring well-structured software (organizational capital than the needs). In Fig. 4(c), it is shown how the intellectual capital becomes higher than the target, i.e. the acquired sum of the ICCs. Thus, the actual performance becomes higher than the objective. In this scenario, it is possible to consider lowering the total intellectual capital or go with having an expected performance that is higher than the objective.

In Fig. 4(b), a scenario is shown where the acquired sum of the intellectual capital is equal to the targeted. Thus, in this situation the intellectual capital matches the needs given through the objective and the task. At the same time, it is important to not only optimize the intellectual capital in relation to the current situation (Fig. 4(b)), but also plan for the future needs. The latter should be captured in the task when it is formulated.

The balancing of ICCs implies that a certain human capital may require a certain level of organizational capital for reaching the target, while the same level of organizational capital may be deemed insufficient for a different human capital. For example, developers with less experience in the software will most likely need better documentation of it than those having worked a long time with the software. Furthermore, social capital plays an interesting role since it may facilitate the development of intellectual capital (Nahapiet and Ghoshal, 1998), for example, good networking with experts outside a unit facilitates learning, the human capital may increase accordingly, and hence increasing the intellectual capital as a whole.

3.5. Theory justifications

The theory is justified through its importance. It provides both practitioners and researchers with a terminology to reason about the relative importance of different ICCs. Furthermore, practitioners may use the theory to profile their units, and to reason about how changes in one ICC may be compensated by improving the same ICC or at least one of the others. Alternatively, it is possible to judge the consequence of changes in intellectual capital profiles using the theory as a basis for reasoning. The theory makes the relationships between ICCs explicit for software engineering.

The theory is based on industrial observations and logical reasoning. As indicated in Section 3.4, it is quite evident that a newcomer to a software development project would rely more on the organizational capital and the expertise of others (social capital), than someone who has been involved in the development of the software over a long period of time. Furthermore, it is no surprise that having a more difficult task and setting, e.g., higher goals in terms of performance (objective) will result in a need for a higher intellectual capital, and thus a higher sum of ICCs than having a very simple task and a lower ambition, e.g., due to that the software is going to be phased out any way.

3.6. Scope of theory

The objective is that the theory is applicable for all types of software development and evolution in which more than one individual is involved. Thus, the theory is not targeting one-person projects or trivial software development. This is also discussed in Section 3.3.2.

The challenges of balancing the ICCs are independent of, for example, the type of software being developed; the development approach used or project constellations (single- or multi-team projects). The theory is hence general for software development and evolution.

3.7. Usage of theory

The theory may be used in several different ways from a practical management perspective. Based on the experience from working with industry (Wohlin et al., 2012) in general, and in particular the research in relation to global software engineering in close industrial collaboration that may be exemplified with the work related to software transfers as reported by Šmite and Wohlin (2012), it is clear that managers in practice do balance different components of intellectual capital. It may not be done explicitly in these terms, but based on experience, expertise and common sense. However, the formulation of the practice as a theory helps managers to make the importance of ICCs and relations between ICCs explicit. The theory systematizes and explicates the common industrial practice, and it will help managers to reason about these issues and also make it easier to communicate the tacit knowledge of an experienced manager. Furthermore, it makes the relationships between different components of intellectual capital explicit so that software engineering researchers better can understand how their research may contribute to industry practice.

The theory will, for example, help managers in relation to answering questions such as:

1. Where are we?
   Managers could reflect on the current performance achievements. By reasoning about the current objective and the difficulty of the task, it is possible to then judge the targeted situation in terms of the different ICCs. Reflection on the sufficiency of the actual ICCs would then explain the performance level. If the situation is not satisfactory actions may be taken, either if being below the target when looking at the sum of the intellectual capital or if being substantially above the target. In the latter case, the manager may choose to pull out some experts to put on another project.

2. Where will we end up (without actions)?
   It is possible to conduct a consequence analysis and reason around the different ICCs. The manager may have a current situation and some change is foreseen or planned, and hence the manager could estimate the consequence of the change. For example, if planning to transfer development from one site to another site, the intellectual capital will most likely change and hence actions may be taken to mitigate this, including improving the organizational capital or moving an expert with the software development for some time (social capital) to work with knowledge transfer (strengthening the human capital). Experiences of similar changes documented according to the theory concepts might help to deal with such consequence analyses.

3. Where do we want to be (what is the target)?
   It is also possible to use the theory to explicate where the development ought to be, i.e., which is the target? The manager may choose different actions to ensure that the target is met. In a given situation with a certain objective and with some tasks at hand, the manager could ensure that there is sufficient intellectual capital to meet the target. The manager could also reason around different alternatives to reach the target, i.e., which ICCs could be most cost-efficiently changed to meet the target?

In summary, the theory makes the tacit knowledge of managers more explicit and it supports managers in their reasoning around the complex challenges related to software development and evolution. The manager is able to reason around the function and dependencies between an objective, a task and the different ICCs, or change the target to something more realistic under the given circumstances. Changing the target may imply either accepting a lower ambition (objective) or simplifying the task if it is deemed impossible to find a cost-efficient solution when it comes to the combination of the ICCs that meets the current target. Furthermore, the theory helps software engineering researchers better understand the relationships between different intellectual capitals and hence put their own research in a larger context.

An illustration of the usage of the theory can be found in Fig. 5. Assume that the organization is prepared to aim for level 3 in terms of the development and evolution levels with the given task (see Section 3.3.1 for the different levels). Level 3 implies that tasks are handled with some effort and occasionally major issues appear that have to be solved. The software developers are not struggling, but they are definitively challenged occasionally. The objective and the task set the target for what to achieve. Given the target, the manager can now look at the intellectual capital available and reason about strengths, weaknesses and different options to reach the target in a cost-efficient way.

In the example in Fig. 5, it can be seen how the manager judge that the software developers have reasonably strong human capital, and the organizational capital is also quite good. The manager judges that the weakest ICC is the social capital. However, in total, the three components should be sufficient to reach the target (shown as perceived intellectual capital). As development goes on the manager may monitor the progress and evaluate whether the judgment is correct. If it turns out that, for example, the organizational capital has been overestimated and in reality the combination of ICCs does not reach the target, the manager now has an explicit mental model of the situation and could discuss actions to address the concerns hopefully more easily (shown as actual intellectual capital). The manager may evaluate different alternatives, i.e. improvement actions, to address the concerns that it seems like the target is not met. The inability to meet the target may show through that it seems like the development is rather on level 2 than as intended on level 3. Thus, the manager may either accept the situation or lower the target, or maybe the development tasks can be changed or the intellectual capital has to be strengthened to ensure that the target is met with the current objective and the development task assigned. Independently, the formulation and illustration of the theory give the manager an explicit framework for conducting a root-cause analysis for performance gaps, reasoning about the balancing of the ICCs, as well as a way of communicating why certain decision are made.

In this section reasoning regarding the usage of the theory is provided, while a practical illustration of how the theory can be observed in many of the decisions taken in a software development project is provided in the next section. The case presented includes a transfer of software development from one development site to another development site within a company, as well as other organizational changes, such as merging two business units, scaling up the number of development teams and distributing development or related components.

4. Empirical case

In this section, a case that illustrates a potential use of the theory and how to operationalize the theoretical constructs and propositions is described.

4.1. Research design

Empirical cases are used in the theory-building process for examination of the validity of theories (Sjøberg et al., 2008). Besides the

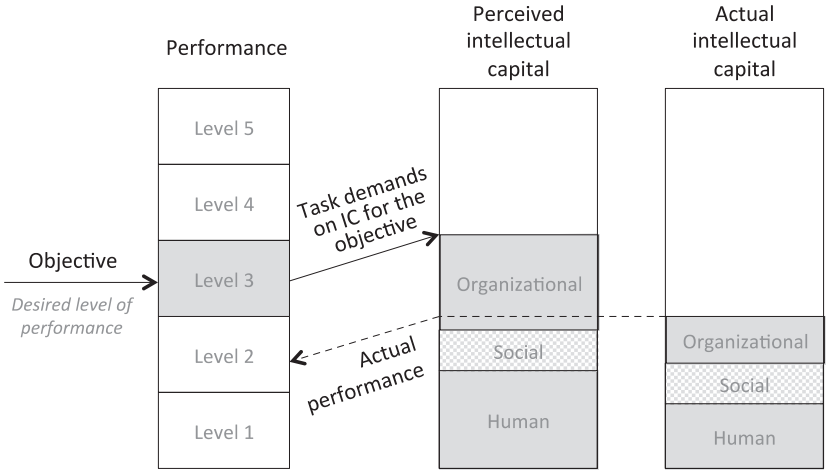

Fig. 5. An illustration of the use of the theory.

validation of the predictive and explanatory powers of a theory, empirical studies can help testing the ability to operationalize the theoretical constructs and propositions. Having said that, validation is best conducted by others to avoid researcher bias, and hence the case presented is focused on the operationalization of the constructs and propositions put forward in the theory. The case study was designed as an exploratory study (Yin, 2009) to investigate the interplay between the human, social and organizational capitals, as well as its relation to the organization's ability to develop and evolve software products. Thus, the case study was not originally designed to illustrate the theory. However, given that the constructs in the theory were used in the case, it became a good case to illustrate the theory as such. In particular, the empirical research was designed to explore the following questions:

- How do developers evaluate their intellectual capital profile of the unit they work in, in relation to the assigned tasks? Are there any events that change the intellectual capital profile during the product evolution?
- How do developers rely on different components of intellectual capital in relation to the assigned tasks? Does it change in different phases of product evolution?
- How do developers perceive their performance in relation to the assigned tasks? Does it change in different phases of product evolution?

The researchers used open-ended questions to explore the phenomena later used in the theory, and sought explanations behind the relationship between the performance and the intellectual capital profiles.

The empirical case described in this article has not been previously reported.

4.2. Context, data collection and analysis

The context of the case study is a multinational software company (below referred to as "the company") and the study object is the evolution of a relatively small sub-system (~100 KLOC) of a compound software system. The sub-system has been transferred from one site of the company to another site belonging to the company—the event investigated as the major event with the strongest impact on the intellectual capital profile of the staff involved in the development. The history of the product evolution is illustrated in Fig. 6.

In this article, the data collected (see Table 4) are used to illustrate the applicability and validity of the theory constructs and propositions. First, individual interviews were conducted with different representatives to capture the questions related to product history, development process and environment, and gathered some observations from visiting the onshore site of the company. In the next step, focus group discussions were held with the Swedish and Indian development teams involved in the evolution to elicit the perceived value of their intellectual capital profiles, reliance on different components of intellectual capital and perceived performance. The unit selected for profiling according to Table 3 was the software teams developing the sub-system (two Swedish teams before the transfer, and two Indian teams after the transfer), and hence their internal collaborative skills are referred to as teamwork skills.

The data generated by the focus groups contained:

- Categorization of ICC aspects from Table 3 (into three groups: strong, medium and weak), which formed an IC profile:
  - Human capital, including Skills and knowledge.
  - Social capital, including Skills of the unit in terms of ability to work together, and External relations.
  - Organizational capital, including Software, Documentation, Organizational culture, and General infrastructure.
- Events that influenced the ICCs, actions that organizations took to balance the ICCs, and consequent changes in the intellectual capital profiles.
- Analysis of reliance on different components of intellectual capital, in which the participants determine the importance of different components (human, social and organizational) during different stages of evolution (before and after the identified events),
- Perceived performance in different stages of evolution (before and after the identified events) using the five performance levels listed in Section 3.3.1.

In Section 4.3, the data gathered in the case study are reported and discussed in the light of the theory constructs.

4.3. Balancing intellectual capital components in practice

Official start of sub-system development: In the beginning, the two Swedish teams that developed the sub-system characterized their intellectual capital (in the group interview) by strong human capital, medium teamwork skills, but weak relations with the stakeholders external to the development teams, and organizational capital with a variety of strong, weak and medium characteristics (see Table 5). In relation to the tasks the performance level determined by the teams as low (level 2—the teams had a hard time to handle the tasks and major problems occurred more often than not). As one of the developers characterized—*"It was a one large chunk of code. It was hard to work with it"*. Since the organizational capital related to the

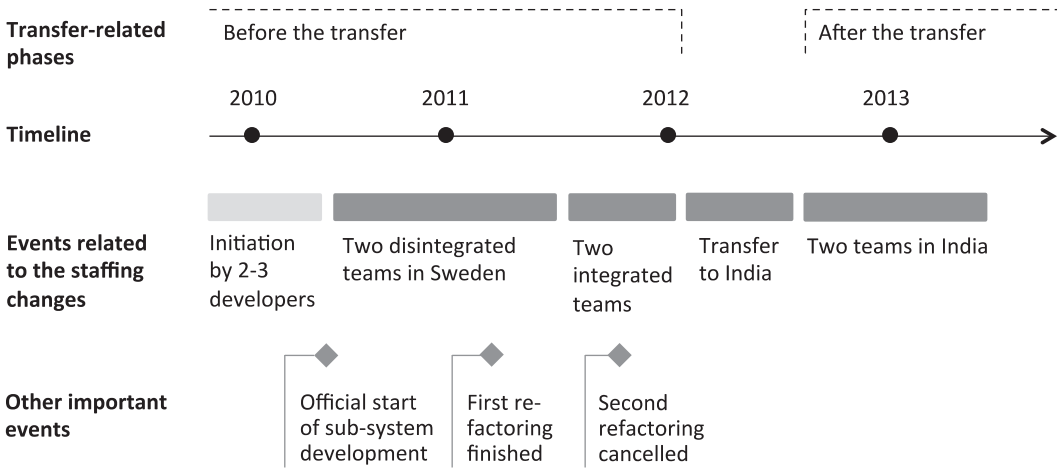

Fig 6. Product history.

Table 4
Empirical data collection.

Method	Number	Duration	Participants	Timeframe
Formal interviews	1	1 h	Transfer manager from Sweden (shortly after the transfer)	October 2012
	6	1.5 h	Swedish developers, an architect, a product owner, a tester	May–June 2013
Survey on knowledge transfer	13	–	Swedish participants (after the transfer)	November 2012
	8	–	Indian participants (after the transfer)	November 2012
Group interviews	1	1.5 h	4 team members from 2 development teams in Sweden	October 2013
	1	2 h	6 team members from 3 development teams in India (conducted via a video-conference)	September 2013
Follow-up interviews	2	1 h	Release manager and a product manager from Sweden	October 2013

Table 5
Starting IC profile of the Swedish teams.

ICC	Categories	Evaluation	Reliance on ICCs	Performance
Human capital	Skills and knowledge	Strong	Human capital	2
	Creativity	Strong		
Social capital	Teamwork skills	Medium		
	External relations	Weak		
Organizational capital	Software	Weak		
	Documentation	Medium		
	Organizational culture	Strong		
	General infrastructure	Medium		

software was weak, external relations of the teams were weak and documentation and teamwork were medium, the developers relied primarily on their skills (human capital), which deemed to be insufficient to reach high performance.

Several mitigating actions were taken to improve the performance as described below (see also Table 6).

First refactoring—Action to mitigate gaps in organizational capital: In order to improve performance, it was decided to organize refactoring, which targeted the source code structure and hence readability, implementation of coding conventions, and a few architectural improvements. Thus the organizational capital increased and the perceived value of these efforts could be observed by the raise in performance from level 2 (the developers have a hard time handling their tasks and major problems occur more often than not) to level 3 (the tasks require some effort and occasionally, major issues may occur, but in most cases, it works quite smoothly). As a developer described: *"It [refactoring] helped a bit but it could have been much more. But there was no time for that. It is still one big chunk of code."* Additionally, more frequent collaboration between the team and the architect (not in the team) was reported as a positive side effect of refactoring.

Integration of the two Swedish teams—Action to mitigate gaps in social capital: The disintegration of the two teams was addressed by organizational changes. Because the teams from the beginning belonged to different managerial structures, there was very little mutual collaboration. Although the teams were located in close proximity (neighbouring workspaces), they received the tasks from their respective product owners, who occasionally had conflicting priorities. The integration of the two teams under one management was done to avoid the coordination overhead. When the teams were united, it had a positive side effect on the collaboration and interaction with other roles outside of the teams. The cooperation between the teams improved and so did the collaboration with the product owners and program managers, who now represented a joint interest. Nonetheless, the teams did not associate these improvements with any significant changes in performance.

Second refactoring—Action to mitigate gaps in organizational capital: The main product architect initiated a second refactoring to improve the software architecture and further improve the performance. Unfortunately, this program was cancelled in the light of the new organizational changes, i.e., a transfer of the sub-system to India, which was a strategic management decision not announced to the development level beforehand. As one manger explained the reason for the cancellation: "*We also started another refactoring that was cancelled because when we heard news that we are transferring we realized that it is better to do the refactoring on the receiving side than doing a lot of changes and stopping half way and handling over half of the work and*

Table 6
Mitigating actions before the transfer.

Actions	Comments	Affected ICCs and aspects (cf. Table 3)		Reliance on ICCs	Performance changes (cf. Table 2)
First refactoring	Action to mitigate gaps in org. capital	+	OC: Source code	Human capital	2 → 3
		+	OC: Software architecture		
		+	SC: Collaboration with experts		
Integration of the two Swedish teams	Action to mitigate gaps in social capital	++	SC: Collaboration with POs and program managers	Social capital	No change
		+	SC: Solving problems together		
		+	SC: Making decisions together		
		+	SC: Shifting workload		
		+	SC: Common goals		
		+	SC: Performance of the unit		
			SC: Give each other feedback		
			SC: Knowing what others are doing		
Second refactoring	Action to mitigate gaps in org. capital	*This refactoring was cancelled due to the subsequent transfer of the sub-system to a new group of developers that did not have enough human capital to finish the refactoring.*			

Notations used for effect illustration: a small increase: +, a large increase: ++, a small decrease: –, a large decrease: ––.

letting them continue". This meant that the original developers who had the human capital to raise the level of the organizational capital did not manage to implement the needed improvements and make it strong, before it was too late. Notably, the teams that received the further evolution of the sub-system were not ready to perform a major refactoring, and thus the improvement program was delayed for several years.

In Table 6, the evolution of product development is illustrated through different events and actions (column 1), changes brought about on the ICCs (column 3), developers' reliance on different ICCs in the light of the events and actions (column 4) and perceived performance (column 5). Additionally, we comment on the intentions of the organization in relation to the events (column 2).

Transfer: Due to a shortage of resources in the Swedish site and new upcoming projects, it was decided to transfer the sub-system to India. A transfer means a relocation of the sub-system from the original developers, i.e., the two Swedish teams, to the new developers, i.e., two Indian teams. After a transfer, new people are working on the product, and hence the level of human and social capital components will change. Thus a transfer means that the intellectual capital profile needs to be updated based on the intellectual capital of the new developers being responsible for the software. The managers anticipated a decrease of the human capital as a result of the transfer, and took preventive actions to compensate the unavoidable gaps. One developer explained why they started cleaning up the code: "*Since we knew about the transfer we tried to clean up these things what we are working with as good as we can. So, it would be easier for others to understand*." Furthermore, it was decided to choose the destination of the transfer to an Indian site within the same company, which already had experience and expertise within the product domain (similar products) and transfers. Employment of developers working on related products resulted in them obtaining important domain knowledge, which ensured a certain level of human capital. Their joint work experience ensured ability to leverage on the social capital in terms of teamwork skills. Some of the Swedes also already knew people on the receiving side in India, which gave a positive impact on the social capital (seen from India). As a system manager explained: "*In this case [a member of a team] knew some of the people already. He had transferred a product to the same site before. … We knew each other. At least on the top level*".

Even though preventive actions were planned the performance decreased. As the architect explained: "*then a lot of new guys came in and they were not doing any work at the start*". To be able to improve fast, the new developers were also supported by improved documentation supporting the product and the processes. While the Swedish developers did not depend on the documentation and thus many documents were outdated, the transfer meant that the organizational capital in terms of the software and documentation would become crucial as the prime source of reliance for the new developers. However investments in documentation, although important, will never be enough to avoid a decrease in performance. As the architect explained: "*Some of the changes they [Indian developers] made affected [a specific feature], for example, and they spent a lot of time fixing [that feature] before they could commit the changes. That of course, is a mistake you make when you are new to the product. You do not really know what you can do and what you cannot do without affecting, for example, performance [of the system]. That comes with the experience. This is something that you really cannot document; you have to get to know the product and how to do the stuff*".

Finally, to ensure the necessary access to expertise a few Swedish developers were partially devoted to support the new Indian teams, and the main product architect was relocated for a half year long onsite support in India to answer questions and act as the safety net for the new team. While this did not raise the human capital in the two Indian teams, it ensured the leverage on the social capital through availability of experts, which helped keeping the level of performance. One Swedish expert explained how he helped out solving problems and tried strengthening the organizational capital: "*They [Indian developers] had some issues […] then I stepped in and helped. Otherwise, I was talking about the next step: what could you do to improve the product and giving them tips for, for example, refactoring*".

As a result of having Swedes available for the Indian teams, the new development unit consisting of two teams in India has the IC profile described in Table 7. Since the two Indian teams received existing software for further development, they had no product knowledge and only medium domain knowledge. Hence the teams primarily relied on the organizational capital. The first months after the transfer the new site climbed the learning curve building their understanding and knowledge of the product (human capital). Interestingly, despite the improvements, the Indian developers perceived the parts of organizational capital related to documentation to be weak and not sufficient to rely on when performing the development work. The transfer evidently resulted in deficiencies in comparison with what was achieved by the Swedish teams, and the resulting performance was on level 2 again.

Several mitigating actions and improvements improved the performance of the new development unit after the transfer (see Table 8).

Assignment of less complex tasks—Action to mitigate task difficulty: To alleviate the problems with performance, the Indian developers were assigned less difficult and critical tasks and more minor product improvements while climbing the learning curve. The

Table 7
IC profile of the Indian teams after the transfer.

ICC	Categories	Evaluation	Reliance on ICCs	Performance
Human capital	Skills and knowledge	Medium	Organizational capital	2
	Creativity	Medium		
Social capital	Teamwork skills	Strong		
	External relations	Medium		
Organizational capital	Software	Medium		
	Documentation	Weak		
	Organizational culture	Strong		
	General infrastructure	Medium		

Table 8
Mitigating actions after the transfer.

Actions	Comments	Affected ICCs and categories		Reliance on ICCs	Performance changes
Assignment of less complex tasks	Action to mitigate task difficulty		No change	Organizational capital	2 → 3
Further investment into documentation	Action to mitigate gaps in organizational capital	+	OC: Product documentation	Social capital	No change
		+	OC: Process documentation		
Gain in working experience	Growth of human capital over time	+	HC: Domain knowledge	Social capital	3 → 4
		+	HC: Software product knowledge		
		+	HC: Creativity		

Notations used for effect illustration: a small increase: +, a large increase: ++, a small decrease: –, a large decrease: —.

Indian teams reflected that this improved their performance to level 3. However, the tasks still required some effort and occasionally, major problems occurred.

Further investment into documentation—Action to mitigate gaps in organizational capital: The deficiencies in documentation were targeted by a continuous improvement program, which facilitated the learning too. The documentation was perceived to be improved from being weak to a medium level. However, this did not have any impact on performance, therefore further improvements were planned. As an Indian developer explained: *"Slowly, slowly we improved the documents. New documents were created too. – Now it is OK, but we need to improve more."*

Gain in working experience—Growth of human capital over time: As the product and domain knowledge grew, the tasks could be handled with less effort. An Indian designer commented in the group interview: *"It is still the same teams. Not a single person has left. The work is interesting. We are growing, we rotate people in different tasks, we are able to balance the workload, and we all sit together"*. The Indian teams said to have gained creativity and performance improved to level 4, when the simple tasks were handled without any major problems.

4.4. Discussion of the illustrative case study

The ICCs and their assessment scales were defined for the case study, as well as the scales for assessing performance. In practice it was observed that the case study subjects could easily understand the concepts, and that ICCs are all relevant for them. Furthermore, the performance evaluation did not cause any difficulty. However, the assessment of the ICCs into strong, medium and weak, and the strength of the impact of certain events on the intellectual capital profile were subject to many questions and disagreements. Thus, how to actually assess the performance should be further evaluated and in particular whether it is possible to measure actual outcome and not only the perception of the participants.

In the illustrative case above several events and actions were studied. A transfer that had a profound negative impact on the intellectual capital profile led to a decrease in performance. The case study illustrates how changes in different ICCs change the performance. Notably, some of the mitigating actions that were implemented to increase the performance through improvements of different aspects of the ICCs did not increase the performance. This means that not all positive or negative changes in the level of intellectual capital will have immediate impact on performance, and that the changes shall be substantial. Notably, the case study also illustrates that the theory has its limitations. There is no mathematical way of calculating the value of each ICC and expressing their combination quantitatively, the theory is unable to clearly explain exactly why certain changes in one or several ICCs are sufficient to improve performance, and why others do not. However, the theory helps to explain and reason about performance after major events (the transfer and changes in task difficulty). It is noteworthy that the theory is on a general level and hence it limits its predictive capabilities. More fine-grained theories and models are needed to be able to make predictions based on actual changes made in a specific context.

Due to the inability to make accurate predictions, the theory is limited in terms of exactness, but it helps in explaining and better understanding the relationships between different key components in the engineering of software. In other words, the changes to ICCs and task difficulty in different contexts should be carefully judged and in the long-term help in improving the predictive power, although the predictive power of the theory will be highly context-dependent.

5. Discussion

It may be observed that the three components of intellectual capital relate to education and research in software engineering as well as organizational specific aspects that cannot be taught directly at the university. In summary, in software engineering, education is primarily focused on the human capital, software engineering research is primarily aimed at organizational capital and the social capital has to be gained by interaction of individuals and to a large extent is influenced by the context in which the individuals develop their professional career.

5.1. Human capital

The main focus of most university education is to increase the human capital of the students and, hopefully, makes students aware of the need for social capital. The human capital is built through courses

at the university as well as the lifelong learning of individuals as humans make a career and obtain different experiences and expertise. Through education and lifelong learning, humans do increase their general experience, knowledge and competence. Specific knowledge, for example, related to a specific domain (such as telecom or process automation), product, system or service to a large extent should be acquired at the workplace.

5.2. Social capital

The social capital is not necessarily taught directly at universities, although students often implicitly become well aware of the need to have good contacts with fellow students and the faculty. Most students leverage on their contact network throughout their studies, for example, by knowing which fellow student to discuss certain courses with and so forth. Furthermore, the social capital is a natural part of project- and team-oriented learning, which is suggested as a complementary responsibility for an educational curriculum. This shall make students understand the importance and the need for social capital when developing software. However, the social capital is very much context-dependent, and hence the social capital is primarily built from the current work. From an educational point of view, the key with respect to social capital is to make students aware of its importance, while their actual social capital will be highly dependent on their future workplace. Notably, training mechanisms exist to improve the social capital of development teams at work. Certain development approaches (such as agile software development) and development practices (such as pair programming, daily meetings and review meetings) foster frequent networking and extensive interaction inside the development teams. Furthermore, communities of practice and participation in different forums foster networking across development teams and units. Organizations that have gaps in human or organizational capital shall take into consideration investments into social capital that can become the source of competitive advantage.

5.3. Organizational capital

The organizational capital is largely addressed by software engineering research, i.e., research targets providing better processes, methods, techniques and tools to support software development. However, the formulated theory implies that software engineering research should take both human aspects as well as social aspects into account. An example of the former is the need for different types of empirical studies with respect to new ideas emerging from research. For example, new tools developed by a PhD student should be properly evaluated by humans and not only proposed, i.e., new tools ought to become part of the human capital and not only a potential organizational capital. Otherwise there is a risk that tools developed as part of research projects end up on the shelf. Thus, the research complements the educational responsibility of a university in a natural way.

6. Conclusions

From empirical observations in industry as described in Section 3.1, it is concluded that industry does balance the components of intellectual capital, i.e. human, social and organizational capitals respectively. In practice frequent changes such as restructuring, retirements, transfers, as well as technical product evolution, continuously challenge the companies' abilities to reach the development objectives and performance. This article packages the observations from industry into a general theory for software engineering. The theory captures the technical aspects of software development through the concept of organizational capital. It acknowledges that software engineering is a human- and knowledge-intensive discipline by including human capital. Furthermore, challenges related to scalability and complexity of software systems make it impossible for a single individual to handle a system of any reasonable size. Development of these systems requires a combination of expertise and experience, and hence interactions between individuals. This is captured in the theory through the inclusion of social capital. The theory could be used by industry to reason about different options when it comes to having a sufficient intellectual capital in a given situation, and by researchers to improve their work in a larger context, i.e. the ICCs. Given the general nature of the theory and the diversity under which software is developed, the theory as such is not aimed at predicting outcomes based on changes in any of the three ICCs. Thus, more fine-grained theories and models are needed to obtain a predictive capability. The proposed theory is focused on understanding, explaining and reasoning about the relationships between human, social and organizational capitals.

It should be noted that the theory emphasizes the importance of intellectual capital in software engineering. It helps to realize that staffing projects is not a straightforward task, and is not only a matter of ensuring individual skills. It is a relationship between the task (its nature and difficulty), and the balance and dynamics between the three ICCs.

The general theory is formulated as a balancing of the three ICCs: human, social and organizational capitals respectively. The theory is firmly based in industry practice, and constructs and propositions have been formulated to structure and systematize the often implicitly handled balancing conducted in industry. The theory helps by providing an explanatory power of the observations in industry, and it may work as a tool to also in general reason about consequences when changing the intellectual capital profile.

Further research is needed in particular others have to test the theory's usability in other settings than those available to the authors of this article, and to find ways to evaluate the theoretical constructs, and hence the theory as a whole. Thus, the further operationalization of the theory still remains. Furthermore, the theory points to the need for software engineering research and education to preferably take all three components of intellectual capital into consideration both when developing new solutions and evaluating them, and when teaching software engineering.

Acknowledgments

We are thankful to Dag Sjøberg for his useful advice that supported us in describing the formulation of the theory. The Knowledge Foundation, Sweden funds the research through the TEDD (Technical Excellence in Distributed Development) Project (grant no. 20120200). The research is also supported by the Smiglo project, which is partially funded by the Research Council of Norway under the grant 235359/O30.

References

Argyris, C., Schön, D.A., 1996. On Organizational Learning II: Theory, Method and Practise. Addison Wesley, Reading, MA, USA.
Adler, P.S., Kwon, S.W., 2002. Social capital: Prospects for a new concept. Acad. Manage. Rev. 27 (1), 17–40.
Bontis, N., 1997. Royal Bank Invests in Knowledge-based Industries 2, 1–4.
Bontis, N., 1998. Intellectual capital: An exploratory study that develops measures and models. Manage. Dec. 36 (2), 63–76.
Bourdieu, P., 1986. The forms of capital. In: Richardson, J. (Ed.), Handbook of Theory and Research for the Sociology of Education. Greenwood, New York, pp. 46–58.
Conway, M., 1968. How do committees invent? Datamation 14 (4), 28–31.
DeMarco, T., Lister, T., 2013. Peopleware: Productive Projects and Teams, third ed. Addison-Wesley Professional, Boston, MA, USA.
Dybå, T., 2001. Enabling Software Process Improvement: An Investigation on the Importance of Organizational Issues. (Dr. ing thesis). Norwegian University of Science and Technology.
Endres, A., Rombach, H.D., 2003. A Handbook of Software and Systems Engineering—Empirical Observations, Laws and Theories. Pearson Addison-Wesley, England.

Faraj, S., Sproull, L., 2000. Coordination expertise in software development teams. Manage. Sci. 46 (12), 1554–1568.
Feldmann, R.L., Althoff, K.-D., 2001. On the status of learning software organisations in the year. In: Learning Software Organizations Workshop. Springer Verlag, Kaiserslautern, Germany, pp. 2–6.
Freeman, P., Wasserman, A.I., Fairley, R.E., 1976. Essential elements of software engineering education. In: Proceedings 2nd International Conference on Software Engineering. San Francisco, USA, pp. 116–122.
Gongla, P., Rizzuto, C.R., 2001. Evolving communities of practice: IBM global services experience. IBM Syst. J. 40 (4), 842–862.
Johnson, P., Ekstedt, M., Jacobson, I., 2012. Where's the theory for software engineering? IEEE Softw. 29 (5), 94–96.
Lehman, M., 1979. On understanding laws, evolution, and conservation in the large-program life cycle. J. Syst. Softw. 1 (1), 213–221.
Moe, N.B., Šmite, D., Hanssen, G.K., Barney, H., 2014. From offshore outsourcing to insourcing and partnerships: Four failed outsourcing attempts. J. Empir. Softw. Eng. 19 (5), 1225–1258.
Mouritsen, J., Johansen, M.R., Larsen, H.T., Bukh, P.N., 2001. Reading an intellectual capital statement: Describing and prescribing knowledge management strategies. J. Intellect. Capital 2 (4), 359–383.
Musa, J., 1975. A theory of software reliability and its application. IEEE Trans. Softw. Eng. 1 (3), 312–327.
Nahapiet, J., Ghoshal, S., 1998. Social capital, intellectual capital, and the organizational advantage. Acad. Manage. Rev. 23 (2), 242–266.
Nonaka, I., Takeuchi, H., 1995. The Knowledge-creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, New York, USA.
Ralph, P., Johnson, P., Jordan, H., 2013. Report on the first SEMAT workshop on general theory of software engineering (GTSE 2012). ACM SIGSOFT Softw. Eng. Notes 38 (2), 26–28.
Rajlich, V.T., Bennett, K.H., 2000. A staged model for the software life cycle. IEEE Comput. 33 (7), 66–71.
Rus, I., Lindvall, M., 2002. Knowledge management in software engineering. IEEE Softw. 19 (3), 26–38.
Schön, D.A., 1983. The Reflective Practitioner: How Professionals Think in Action. Basic Books, New York, USA.
Schultz, T., 1961. Investment in human capital. Am. Econ. Rev. 51 (1), 1–17.
Sjøberg, D.I.K., Dybå, T., Anda, B.C.D., Hannay, J.E., 2008. Building theories in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (Eds.), Guide to Advanced Empirical Software Engineering. Springer Verlag, Heidelberg, Germany, pp. 312–336.
Šmite, D., Wohlin, C., 2010. Software product transfers: Lessons learned from a case study. In: Proceedings of the International Conference on Global Software Engineering. IEEE Computer Society, Princeton, USA, pp. 97–105.
Šmite, D., Wohlin, C., 2011. Strategies facilitating software product transfers. IEEE Softw. 28 (5), 60–66.
Šmite, D., Wohlin, C., 2012. Lessons learned from transferring software products to India. J. Softw.: Evol. Process 24 (6), 605–623.
Stewart, T., 2001. The Wealth of Knowledge: Intellectual Capital and the Twenty-First Century Organization. Nicholas Brealey, London.
Tobin, J., 1969. A general equilibrium approach to monetary theory. J. Money Credit Bank. 1 (1), 15–29.
Willcocks, L., Hindle, J., Feeny, D., Lacity, M., 2004. IT and business process outsourcing: The knowledge potential. Inform. Syst. Manage. 21 (3), 7–15.
Wohlin, C., Aurum, A., Angelis, L., Phillips, L., Dittrich, Y., Gorschek, T., et al., 2012. Success factors powering industry-academia collaboration in software research. IEEE Softw. 29 (2), 67–73.
Wohlin, C., Šmite, D., 2012. Classification of software transfers. In: Proceedings 19th Asia-Pacific Conference on Software Engineering (APSEC). Hong Kong, pp. 828–837.
Yin, R.K., 2009. Case Study Research: Design and Methods, fourth ed. Sage, Thousand Oaks, CA.
Youndt, M., Subramaniam, M., Snell, S., 2004. Intellectual capital profiles: An examination of investments and returns. J. Manage. Stud. 41 (2), 335–361.

Claes Wohlin received the Ph.D. degree in communication systems from Lund University in 1991. Currently, he is a professor of software engineering and dean of the Faculty of Computing at Blekinge Institute of Technology, Sweden. He has previously held professor chairs at the universities in Lund and Linköping. His research interests include empirical methods in software engineering, software process improvement, software quality, and global software engineering. He was the recipient of Telenor's Nordic Research Prize in 2004, and a member of the Royal Swedish Academy of Engineering Sciences since 2011. He is editor-in-chief of Information and Software Technology published by Elsevier.

Darja Šmite received her Ph.D. degree in computer science in 2007 from the University of Latvia. Currently, she is an associate professor of software engineering at Blekinge Institute of Technology in Sweden, where she leads the research efforts related to the effects of offshoring for Swedish software-intensive companies. She is also a visiting professor at University of Latvia. Her research interests include global software engineering, large-scale agile software development, and software process improvement.

Nils Brede Moe works with software process improvement, agile software development and global software development as a senior scientist at SINTEF Information and Communication Technology. His research interests are related to organizational, socio-technical, and global/distributed aspects. His main publications include several longitudinal studies on self-management, decision-making and teamwork. He wrote his thesis for the degree of Doctor Philosophiae on "From Improving Processes to Improving Practice —Software Process Improvement in Transition from Plan-driven to Change-driven Development". He is also holding an adjunct position at Blekinge Institute of Technology.