Supporting Joint Idea Generation with Software Prototypes in Offshore-Outsourced Software Development Projects

Maike Winkler
University of Bern
maike.winkler@iwi.unibe.ch

Thomas Huber
University of Bern
thomas.huber@iwi.unibe.ch

Jens Dibbern
University of Bern
jens.dibbern@iwi.unibe.ch

Abstract
Joint idea generation is vital in software development projects requiring team members with different knowledge specializations to exchange and integrate multiple perspectives into ideas to improve the software product. While joint idea generation is generally difficult to achieve, it is even more challenging in offshore-outsourced settings. Our goal was to understand the process of how software prototypes can support joint idea generation over the life of a 16 month offshore-outsourced software development project. Based on an in-depth, ethnographic case study, we reveal three joint idea generation modes building on and stimulating each other: from diverging, to exploring and advancing. These joint idea generation modes were closely interwoven with the software prototype. We find that as the software prototype evolved, new possibilities for engaging in various joint idea generation modes emerged. Our research has important implications for literature and practice.

1. Introduction
Today, successful software development projects require ideas and solutions that draw on the diverse knowledge and perspectives of multiple actors [e.g., 1, 2]. Diverse knowledge bases offer possibilities to leverage each other’s distinct knowledge specializations to jointly discover various solutions and produce more innovative and synergistic ideas [3]. When participants do not share ideas, this can lead to failures of IS projects or poor project performance [3-5]. Thus, there is a considerable need to engage in joint idea generation, i.e., the process in which team members with different knowledge specializations exchange and integrate multiple perspectives into ideas with the goal to improve the software product. While joint idea generation is generally difficult, it is even more challenging when software projects are offshore-outsourced. In particular, the often persistent problem of “vendor silence” hinders client and vendor from realizing the potential of their diverse knowledge specializations [e.g., 4, 5, 6]. This is often due to status differences which set client and vendor apart. Accordingly, clients dominate the joint task and vendors neither question ideas nor engage in discourse [3]. What is more, client and vendor often do not share common knowledge [7] and their distributed nature limits opportunities to exchange knowledge [8]. Vendor employees are often completely unfamiliar with the desired software product at the outset of a project.

However, nowadays client and vendor employees often interact with each other mediated by the software prototype - particularly when agile software development methodologies are applied [9]. As such, team members interact by co-generating, adjusting, and refining the prototype [e.g., 10]. Our overarching idea is that this supports joint idea generation and thus helps overcoming the above outlined challenges. While joint idea generation is important at any point in time during the software development process [1], we suggest that the ways in which joint ideas unfold are likely to differ at different points because joint idea generation builds on conditions that emerge over time. For example, certain practices may not be possible in early stages of the development process in which the representation of the prototype is rather rough; yet, once the prototype matures, different ways of joint idea generation may be possible. We suggest that as the prototype evolves over time, joint idea generation co-evolves. Since we examine this emergent phenomenon unfolding through everyday interactions between client and vendor, it was important to conduct a longitudinal, process-oriented study. Our study seeks to answer the following research question: How can software prototypes support joint idea generation in client-vendor interactions over the life of an offshore-outsourced software development project? The paper is structured as follows. In a first step, we present the theoretical background and research framework for this study. In a next step, we elaborate on our method and demonstrate the findings of our empirical study. Finally, we discuss the contributions and implications for theory and practice.
2. Theoretical background

2.1. Joint idea generation

Prior innovation scholars emphasized that idea generation can be understood as part of a process with certain stages through which innovation unfolds [e.g., 11, 12, 13]. Idea generation is often associated with exploration in which team members engage in search, experimentation and discovery to create novel ideas [14, 15] and is needed throughout the entire innovation process in order to be able to respond to unpredictable problems or tasks and create opportunities [16]. Realizing that diverse actors work together in teams, organizational scholars started to examine how ideas are generated collectively [17, 18]. Yet, they focused on co-located settings and paid little attention to the various challenges in distributed settings, especially when development projects are offshored. Recently, offshore-outsourcing scholars have become increasingly interested in the role of joint activities for project success [19]. For example, in successful offshore projects, client involvement facilitates joint problem solving [20]. Joint idea generation differs from joint problem solving as there does not necessarily need to be a problem to solve. Especially in agile software development projects, features may be added, modified or deleted at any time [9] requiring proactive discussions to generate ideas throughout the whole process. Joint idea generation goes beyond the sharing of an idea towards a discourse between actors which allows the integration of each other’s perspectives. In this way, the concept overlaps in many ways with what Levina and Vaast [3] refer to as collaborative effectiveness, i.e., diverse actors share different perspectives with each other, the actors listen to and understand their perspectives and ideas are included in the development process (when appropriate) in mutually beneficial ways. While prior literature revealed valuable insights on joint idea generation, less attention has been paid to the role of artifacts in supporting the process of joint idea generation. Yet, this is vital in distributed offshore teams since team members heavily rely on artifacts which mediate their interaction.

2.2. How joint ideas unfold when artifacts are used

Joint ideas are created in iterative dialogues in which interactants take different perspective and form novel interpretations [e.g., 21, 22]. Since knowing is a situated and contextual activity which is “enacted in the moment” [23 p.253-3], ideas are generated in particular situations (e.g., software task). As soon as the situation changes, an idea may not be relevant or useful anymore. From prior literature we know that artifacts can support such dialogues to integrate viewpoints and stimulate knowledge creation [e.g., 24, 25, 26]. Artifacts may operate as boundary objects, i.e., they “adapt to local needs and the constraints of the several parties employing them, yet robust enough to maintain a common identity across sites” [27 p.393]. They serve as a common reference point to enable work across actors with different knowledge specializations [28]. Simultaneously, their interpretive flexibility allows team members to read the artifact in various ways and from different perspectives to create and attach meanings and interpretations to them [29, 30]. Further, artifacts may operate as epistemic objects, i.e., they embody “at once by what they are and what they are not (or not yet)” [31 p.5-6]; they “are as much defined by what they are not (but will, at some point, have become) than by what they are” [32 p.90]. Their dynamic nature stimulates knowledge creation in different situations and contexts over time [31]. Next, we describe how software prototypes as key artifacts in software development support joint idea generation.

2.3. Software prototypes as central artifacts in software development

Nowadays, agile software development methodologies are increasingly applied in offshored settings [e.g., 33]. In those settings, a running software prototype is frequently developed in short iteration cycles and continuously used to support communication across team members [9]. We define software prototypes as consisting of pieces of software which are already developed and functioning (e.g., an operating functionality) as well as not yet developed software pieces (e.g., drawings, written requirements) [34]. We believe that its role in supporting joint idea generation during client-vendor interactions over time is vital for several reasons. First, the software prototype is common enough to enable and mediate interactions between client and vendor. It represents a “current state” artifact that can be used as reference point to convey knowledge to each other [27]. While it represents the current, yet necessarily provisional software prototype – at the same time it represents the desired product of the software development project. Second, the software prototype is flexible enough to be interpreted from different perspectives [27], i.e., client and vendor can interpret the same representation in the software prototype in various ways. The representations may also be experienced differently by the same individuals at different points in time and in different contexts [35]. Third, the software prototype is frequently becoming over time [10]. As such, it is only a partial representation
of the end product and “always in the process of being materially defined” [32 p.90] to discover novel possibilities and directions [36]. Thus, changes in the prototype may enable new conditions for practices which were not possible at earlier points in time. To date, the software prototype as a dynamic object for joint idea generation has remained largely unexplored.

3. Method

In this research, we conduct an in-depth case study [37] and follow a process-oriented approach to explain the emergence of joint idea generation. Taking a process perspective helps us to capture and explain in-depth when, how and why changes occur over time [38]. The data discussed in the following paragraphs draws on an ethnographic case study of an offshore outsourced software development project. Good ethnographies involve direct, rich and continued observations of social interactions to capture everyday life and practices, beliefs and lived experiences [39]. Hence, we entered the field as non-participant observers to capture and interpret client-vendor interactions based on very rich and longitudinal data sources covering an observation period of 16 months.

3.1. Case context

An entrepreneurial client located in Switzerland had a vision for a new software tool which we refer to as NovaProduct. The novel software tool should enable users to manage, organize and view information in new ways. In particular, users should be able to create and share compressed content as well as order and view content in different ways. The single client had no knowledge on what was technically possible. He hired a small offshore development team in Vietnam consisting of three software developers, two Scrum Masters and a third-party designer. All offshore team members were employed by a small vendor company and situated in Vietnam; the designer was a freelancer. Joint idea generation was very important to make continuous improvements to the software product. As the team applied agile software development methodologies, ideas for functionalities, features or designs were constantly added, modified or removed. These constant changes required team members to continuously engage in joint idea generation throughout the whole development process. Client and vendor employees had different knowledge specializations. The client created the software vision for several years without a background in software development. In contrast, the vendor had strong technical knowledge; yet, developers were completely unfamiliar with the desired software product. This made joint idea generation particularly difficult. Agile work practices ensured the centrality of software prototypes and client-vendor dialogues. Client and vendor employees could engage in dialogues during frequent weekly meetings. It was common that all developers and Scrum Masters attended the meeting, except the designer. Dialogues between team members were supported and mediated by the use of the software prototype. In particular, developed or not yet developed functionalities, features or designs were made visible to each other via screen sharing.

3.2. Data collection

To answer our research question, we collected data from three different sources. Our primary data source was recorded observational data: We attended and recorded all virtual (video and audio) meetings between the vendor employees and the client and took additional field notes. These meetings took place between 2-3 times a week over a 16 month period. During these interactions, client and vendor shared the latest prototype and had the opportunity to discuss improvement needs and ideas with each other. In total, we attended 145 meetings which resulted in 57 hours of observation and 455 handwritten pages of observation notes. Furthermore, we conducted 16 in-depth interviews in three rounds (I1: Round 1; I2: Round 2; I3: Round 3). We recorded and transcribed a total of 16 interviews ranging from 30-90 min. We conducted skype interviews with all project participants. Interviews with some of the developers took place in written form due to language barriers. The interviews focused on obtaining a retrospective view on the development process and changes that occurred. The above-mentioned data sources were complemented by log files and the content exchanged via collaboration system: All log files and content exchanged via the collaboration tool were tracked. This collaboration tool included comprehensive information on project documents including all requirements, written discussions, plans or changes. This allowed us, for example, to capture the details about requirements and which functionalities, features or designs were developed at which points in time.

3.3. Data analysis

Stage 1 - Identifying instances of joint idea generation: In a first step, we identified those instances in which joint idea generation occurred. We refer to an instance of joint idea generation when the following conditions were met: Instances always involved a dialogue between client and vendor, the use of the software prototype during such dialogues and an idea.
For example, the client suggested an idea and used the software prototype to discuss the idea with the vendor; in return, the dialogue supported by the software prototype helped the vendor to contribute with own inputs. Such instances did not include situations when the client explained an unclear requirement to the vendor. The instances of joint idea generation represented our unit of analysis and consisted of three main elements (Figure 1). These elements did not follow a particular sequence.

**Figure 1. Elements of a joint idea generation instance**

We carefully reviewed all notes taken during our observations and listened to recordings of the virtual meetings between client and vendor employees to determine all instances of joint idea generation in which software prototypes were used. This helped us to identify a total of 50 instances. Then, we used the recordings of the virtual meetings to transcribe these dialogues, and prepare rich descriptions on how the prototype was used during each instance in an Excel sheet.

**Stage 2 - Analyzing instances of joint idea generation:** The goal of this stage was to explore how client and vendor employees engaged in joint idea generation when software prototypes were used during interactions. Based on Glaser and Strauss [40] and Miles and Huberman [41], we used grounded theory methodology, started to develop initial categories of joint idea generation and analyzed how the evolving software prototypes supported these. The analysis was an iterative process of going back and forth between the data and theory. During the coding process, we continued to be open towards new aspects emerging from our data [42]. We coded how ideas were generated during joint interactions and then grouped them to similar concepts which were then abstracted into three higher-level categories with explanatory power [42]: Diverging, Exploring, Advancing (Table 1). We refer to them as joint idea generation modes. Each mode includes a dialogue and prototype use to generate an idea. In addition, we used recordings of virtual meetings, collaboration tool data and field notes to analyze the characteristics of the software prototype.

**Stage 3 - Comparing and Theorizing:** During this stage, we divided the data into successive time periods (temporal brackets) to reveal changes in joint idea generation across time periods. We determined a shift in time periods when project dynamics changed (e.g., team member change) and distinguished 4 time periods (PI-PIV). PI covered the initial 3 months of the project followed by PII which lasted 3 ½ months. Subsequently, PIII encompassed 3 months and PIV lasted 6 ½ months. We systematically compared joint idea generation modes with software prototypes characteristics over the project’s life to identify recurring patterns [38]. The bracketing of data into temporal episodes “enables the explicit examination of how actions of one period will affect action in subsequent periods” [38 p.703]. The analyses were conducted by the first author. To ensure validity of our analyses, we heavily triangulated the three different data sources.

### 4. Findings

Our findings show that the software prototype was closely interwoven with different modes of joint idea generation which emerged and unfolded over the life of the software development process: from diverging, over exploring to advancing. We provide a brief overview of these modes in Table 1.

**Table 1. Modes of joint idea generation**

<table>
<thead>
<tr>
<th>Diverging</th>
<th>Exploring</th>
<th>Advancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using basic prototype to discuss different perspectives - scrutinize other’s ideas - defend own viewpoints</td>
<td>Using enhanced prototype to - trigger and answer new questions - discover new possibilities or considerations</td>
<td>Using mature prototype to - build on each other’s inputs - complement contributions - connect ideas</td>
</tr>
</tbody>
</table>

These modes of joint idea generation represent analytically different patterns. We illustrate joint idea generation modes in client-vendor interaction for each time period. Table 2 reveals how often each mode was present individually or in combination in each period and over the course of time periods.

**Table 2. Counts of joint idea generation modes**

<table>
<thead>
<tr>
<th>Modes</th>
<th>PI</th>
<th>P II</th>
<th>P III</th>
<th>P IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>D, E, A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>xxx</td>
</tr>
<tr>
<td>A, E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>xxx</td>
</tr>
<tr>
<td>D, A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>xx</td>
</tr>
<tr>
<td>D, E</td>
<td>-</td>
<td>-</td>
<td>xxx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>A</td>
<td>-</td>
<td>-</td>
<td>xxxxxxx</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>-</td>
<td>xxx</td>
<td>xxxxxxxxx</td>
</tr>
<tr>
<td>D</td>
<td>-</td>
<td>xxxxxx</td>
<td>-</td>
<td>xxx</td>
</tr>
</tbody>
</table>

D: Diverging, E: Exploring, A: Advancing, x: Instance of joint idea generation
4.1. Period I: No joint idea generation

Within this initial time period, team members worked on the rough NovaProduct functionality and design. Tasks were loosely assigned and planned. The prototype at this early stage incorporated very few, mostly inactive and incomplete features. The prototype allowed a look at the general three-column screen design and the creation of initial content. Client and vendor employees did not engage in joint idea generation. While the client had a vague vision of what NovaProduct should look like, the developers had very little knowledge of the desired end product, i.e., they did not know what they were developing.

“...they see nothing at the beginning...The client has an idea, has a vision, that’s what he wanted ... there is nothing, except the idea.” [SM1, I2]

In addition, the developers applied novel technologies in this project which they have not used before. Thus, they also had to spend time to become familiar with the novel technology. It was the responsibility of the client to think about the software product and vendor employees followed and relied on his ideas.

“It is his vision, I think the man with the vision knows how the vision should be done...” [TPD, I1]

During two weekly virtual meetings vendor developers primarily engaged with the client to understand the client’s ideas. The vendor was reluctant to discuss ideas at this early stage while the client attempted to keep his ideas uninfluenced.

4.2. Period II: Diverging

In this time period, the team focused on developing the basic NovaProduct functionality and design. They planned to develop a certain amount of tasks every two weeks. At this stage, the prototype had more interactive functionalities. It was then possible to share content across individuals and groups. Different types of content could be stored (videos, text, photos etc.). The basic prototype also allowed viewing content from different point of views. For example, changing the view on content from “private” to “public.” Using the representations in the software prototype at this point in time helped to make ideas “tangible” and “feel real” [C, I1]. During this time, the vendor still had limited knowledge on the client’s ideas and the overall software product; thus, developers were likely to produce different interpretations on how the software could be improved than the client.

Within the scope of this time period, client and vendor started to engage in a joint idea generation mode that we refer to as diverging. By diverging we mean that client and vendor produced different viewpoints and perspectives on how the software could be improved. They used the prototype to scrutinize, defend and discuss different perspectives. A number of instances of joint idea generation through diverging became evident. For instance, the vendor suggested an idea: A developer realized an issue for improvement for a not yet developed feature. The developer opened a table in a requirement displaying an overview of icons to change the view on content in a column. He then pointed to a particular set of icons to articulate his concerns: From his perspective, as all icons were represented in the same blue color, this created the impression that all icons could be clicked on, yet one icon was not clickable, i.e., follower icon. He referred to another set of icons displayed in the same table but in different colors to suggest an idea for improvement: he suggested that changing the color of icons would make it clearer which icons a user can click on and which not.

Vendor: But ...I cannot know which button I can click. In this case all are blue so I can click any icon, but the follower icon is not clickable.
Client: Yes, you are right.
Vendor: Yes, it makes the user confused ... we only make the public icon in blue, follower in that [pointing to a grey color], because it is not clickable...

After the vendor suggested this idea, the dialogue between vendor and client began. At this point in time, icons to change the view on content were already developed for Novs, but not yet for the whole column. They used the interactive nature of this similar feature that was already developed (clicking on icons) to compare it to the not yet developed feature and discussed their different perspectives. This experience of the current software prototype helped the client to scrutinize whether the developer’s idea would really improve the user experience. He used the representation to defend his idea of keeping the blue color for all icons and let users try and find out through the actual clicking activity which icons were clickable. From his perspective, there was no need to change the color of icons. Thus, the basic, interactive functionality supported client and vendor to produce different viewpoints and perspectives and to discuss these.

Client: Yes, but the problem for me is that this is not really correct...it's not representing what’s happening really.
Vendor: Mhm.
Client: ...I think a user gets familiar with it if he starts clicking and try out. But then he can see clearly that we have different states. For me it is clearer like this.
Vendor: Okay, it is up to you.

The vendor often expressed that the final decision which of the different ideas should be chosen needed to be made by the client. The vendor was cautious and less
comfortable to represent own ideas during this early stage. In the following months, client and vendor enacted additional joint idea generation modes.

4.3. Period III: Exploring

During this time period, the team mainly focused on enhancing existing functionalities and features as well as developing new ones. They continued to plan a certain amount of tasks for every two weeks. At this stage, the software prototype encompassed a wider range of developed, functioning elements allowing more possibilities for sharing and viewing diverse types of content (e.g., pdf). In addition, it was possible to receive updates on other users activities and administrative support. The software prototype was then also opened to friends and acquaintances of the client as initial users as part of a closed beta test. At this point in time, developers had gained more knowledge on the client’s ideas through the frequent interaction with the prototype and the client; yet, they still had difficulties to grasp the overall software vision.

Within the scope of this time period, client and vendor started to engage in a joint idea generation mode that we address as exploring in addition to diverging. We refer to exploring when client and vendor used the software prototype during interactions in a way that triggered new possibilities, questions and considerations which they did not think of before; this often included a subsequent discussions to answer such newly arising questions. Exploring ideas was evident in several instances. For example, the client suggested an idea: He realized that it was not possible to scroll content entered in a Nov. A Nov served as an information carrier to enter and store compressed information. The client then suggested to limit the number of characters that should be typed in a Nov. Then it would not be necessary to scroll content. The vendor added the client’s idea to the software requirement.

Client: I saw that...we can't scroll text in a Nov. There are no arrows anymore, perhaps I thought it would be better if we limit the number of characters allowed to be typed for a user on a Nov. Because we don't really need to add a huge text to a Nov.
Vendor: Yes.
Client: So, perhaps we could include this in this ticket.

After the client suggested the idea, the dialogue with the vendor began while both of them looked at the client’s idea which was added to the requirement. This raised a new question which remained to be explored. The developer wondered how many characters were needed. In return, this triggered new considerations about the space in the database. To answer the new question, they then opened the current software prototype. At this point in time, the design of Novs was enhanced with more elements including fields for entering a title, picture, description, arrows to change the page as well as information on usage and activity. This helped client and vendor to imagine and estimate how much text would make sense by looking and experiencing the available space for entering a text in the description field. The developer used the representation in the software prototype as a reference point to count lines in the description field which helped him to suggest an idea which answered the new question: to limit the characters to four lines.

Vendor: Yes, but we need to check how much in the database. So, the question is about how many characters here are okay for you?
Client: Mhm...
Vendor: One, here, one two three, four...four lines or something?
Client: Yes...
Vendor: Four lines...yes, okay!

Often “diverging” initiated “exploring” in an instance of joint idea generation. For example, after client and vendor employees scrutinized, defended and discussed diverse viewpoints to generate different ideas, this then triggered novel questions and considerations. Over time, client and vendor enacted additional joint idea generation modes.

4.4. Period IV: Advancing

At this point in time, the project focus shifted to prepare the software prototype for end users. Only the main developer worked on the project with the client. After two months, a developer re-joined the team. They planned and assigned tasks loosely. At this stage, the software prototype encompassed mature functionalities, features and designs. The prototype allowed different forms of interactions across individuals and groups (e.g., chat, messenger), the manner in which features could be used was adjusted (e.g., how to control the view on content) and more details were added. The mature prototype helped developers to “visualize the whole picture” [D2, I3] to better understand how requirements were connected with each other and form the software vision. This made it easier for them to create ideas. The vendor has become increasingly able, comfortable and confident to share and discuss ideas with the client. For example, in some cases a developer was confident enough to actually develop his idea in the software prototype before discussing the idea with the client.

Within the scope of this time period, client and vendor gradually started to engage in advancing in addition to diverging and exploring. We refer to advancing when client and vendor used the software
prototype to build on each other’s inputs, connect ideas or complement each other’s contributions. The software prototype helped them to unite and integrate contributions and inputs such that these were merged and melded during the process. This mode of joint idea generation emerged in several instances. For example, the vendor suggested an idea: A developer recognized an issue in the current software prototype that required a solution. As functionalities were very mature and detailed, there were a lot of possibilities for adding content to a Nov. At this point in time, each Nov had several buttons on the right which could open a new field (e.g., Nov chat). Yet, the issue was that content was often scrolled in the whole column instead of within the field of a Nov. The developer showed this issue in the current software prototype and then suggested the idea to lock the middle column once the user opens a new field by clicking on a button on the right of a Nov.

Vendor: Because of the limitations of the browser and technology, we cannot... It is difficult to lock the scrollbar... even if we scroll the item in this item. If we move [the] mouse and scroll to the end point of this item, it will scroll item...[in] the middle column. So, I’m thinking about a solution... we should lock the screen of the middle column.

After the vendor suggested the idea, the dialogue with the client began. The detailed representation of the software prototype helped them to build on each other’s input. The mature prototype was not only detailed for Novs, but also displayed the whole column with several Novs. This comprehensive representation helped the client to build on the vendor’s idea: he referred to the mature prototype to state that, in addition to keeping the scrolling in the middle column locked, it needs to be possible to click on other Novs displayed in the middle column. The mature design and functionality provided a full picture of the desired software product that supported client and vendor in advancing ideas.

Client: Perhaps, yes, yes. Lock the middle column.
Vendor: But if we do it that way, we have to close the field again, for scrolling.
Client: Is it possible to have the screen locked, but to see if a user is clicking another Nov? So that only... disable the scrolling, you know. But not the clicking. So if the user clicks another Nov it feels like not locked screen.
Vendor: Yes, yes I see...

Client and vendor increasingly used the software prototype in ways that the different joint idea generation modes stimulated each other. For example, client and vendor discussed different viewpoints which triggered new considerations, in return, these were then used to build on each other’s ideas.

4.5. A Process model of joint idea generation

Together, the three joint idea generation modes through which ideas emerge form a process model (Figure 2). The process model proposes that certain joint idea generation modes will emerge over a project’s life. Our findings suggest that the evolving software prototype characteristics were closely interwoven with the joint idea generation modes and facilitate their emergence over the project’s life. When the prototype was very rough at the outset, the few inactive and incomplete features did not support client and vendor to jointly generate ideas. Over time, as new features were added, conditions of how the prototype could be used changed and allowed different ways of joint idea generation. Some basic, interactive features were necessary to engage in diverging, yet, these were not sufficient to enable exploring or advancing. At the same time, team members gained knowledge through the interaction with each other and the evolving prototype. Thus, while the software prototype was a major driver of change, other context factors such as knowledge, project focus and way of work impacted and were impacted by such interactions. Context factors, prototype and joint idea generation were intertwined within each time period and co-evolved across time periods. For example, the knowledge obtained when diverging helped team members to engage in different ways of joint idea generation in the next time period. When the prototype had extended and enhanced functionalities and designs, it allowed for more possibilities to engage with a wider range of functioning elements. Then, it was also possible to explore. Once the prototype was mature, the detailed, fully interactive functionality and design enabled client and vendor to engage in advancing. The different joint idea generation modes not only unfolded over time, they also built on and stimulated each other to fully integrate multiple perspectives into ideas to improve the software product.

5. Contributions and Implications

Our goal was to reveal how the software prototype supports joint idea generation over time. From our qualitative analyses, our results provide important contributions and implications to outsourcing literature and valuable guidance for practitioners. Due to the extreme challenges in offshore settings, joint idea generation is particularly difficult at the outset of a project and unlikely to take place in the initial months. Our findings show that possibilities to engage in various ways of joint idea generation over time depend on the characteristics of the software prototype. We reveal how the process of joint idea generation unfolds by
identifying three joint idea generation modes which emerged over time: *diverging, exploring* and *advancing*. The process model (Figure 2) increases our theoretical understanding of when, how and why joint idea generation emerges in offshored software development.

When the prototype is at the very early stages, the rough, mostly inactive and incomplete functionalities and design do not support joint idea generation. While team members get a feeling for what the software should look like, fruitful discussions on how to improve the software product are not encouraged. Yet, as prototype characteristics change and become more interactive over time, joint idea generation is possible even in offshore settings. Team members can then use already developed functionalities as a reference point to compare it to not yet developed ones to discuss ideas. Such shared practices facilitate collaboration [43] and vendor employees can obtain more knowledge on the client’s ideas to contribute and discuss ideas. Once the prototype is extended and enhanced with a wider range of functioning elements, additional ways of joint idea generation are possible. The prototype then offers more possibilities to use it as a reference point for making more exact estimations. Yet, the prototype needs to have enough fully interactive and detailed features to enable team members to fully integrate their perspectives. Such detailed, comprehensive range of features allow a complete picture of what the software should do and help visualize the whole, overall vision of the software product. The different modes not only occurred individually, but stimulated each other to enable combinations of modes. Our findings suggest that as different joint idea generation modes are possible, the generated ideas become more refined over time. This complements prior offshore literature [3, 5] by revealing a new relationship between particular modes of joint idea generation and the maturity of a generated idea at certain points in time.

Our findings add to prior literature by focusing on the evolving prototype which allowed team members to draw on a greater range of joint idea generation modes to integrate multiple perspectives into ideas to improve the software product. Prior studies on joint idea generation did not consider the prototype, yet, recognized that it is often not enough to share documents and project material to enable team members to fully contribute their expertise. While rather incomplete representations may be suitable to inspire joint ideas in co-located teams [e.g., 31], they do not facilitate joint idea generation in offshored settings. We unveil the conditions under which prototypes can facilitate joint idea generation. Our findings show that the software prototype needed to have enough fully interactive, complete and detailed range of elements such that client and vendor employees could use it to fully integrate their multiple perspectives. This adds to offshoring literature that emphasized the need for managers to induce vendors to fully contribute and facilitate open dialogues [e.g., 3, 4, 5]. For example, they could draw on certain resources to ensure that vendor employees had enough status to fully contribute [3]. The prototype represents a resource which both client and vendor employees can access and use at any point in time as it evolves.
stage of the development process to facilitate joint idea generation.

Moreover, our study has important implications for practice. Offshore project members need to be aware that different possibilities for joint idea generation emerge as the characteristics of the prototype evolve. It is important that they consciously react to such changes in the prototype. When producing software products, practitioners need to consider factors such as the level of completeness, interactivity and range of features at certain points in time. They should make sure that the prototype is current as well as accessible [28] and shows the characteristics identified in this study. While it is unrealistic that the prototype can be developed in a way that it shows these characteristics much earlier, sophisticated prototyping tools could enable a better interaction without extensive efforts. Over time, they can more purposefully engage in various ways of joint idea generation to improve the software product.

5.1. Limitations

Our study has several limitations. First, we conducted a single case study of an agile offshore-outsourced software project with a single client based in Switzerland and a small offshore team situated in Vietnam. Future longitudinal studies are needed to examine the dynamics of joint idea generation over time in non-agile development projects. In addition, we focused on changes of the software prototype influencing joint idea generation; while we identify other context factors, future studies could examine more closely how these factors are intertwined with joint idea generation modes. Further, scholars identified that cultural differences influence vendor silence [e.g., 5]. While interactants have different cultural backgrounds in our case study, further research is needed to take into account how these cultural differences may have influenced joint idea generation. While our findings focus on joint idea generation between client and vendor in offshore-outsourcing settings, our research focus differs from other studies that looked at how distributed team members who were purposefully brought together used digital tools to innovate [e.g., 44].

6. References


