Generalizing by Similarity:
Lessons Learnt from Industrial Case Studies

Smita Ghaisas\textsuperscript{a}, Preethu Rose\textsuperscript{a}, Maya Daneva\textsuperscript{b}, Klaas Sikkel\textsuperscript{b}, Roel J Wieringa\textsuperscript{b}
\textsuperscript{a}Tata Research Development and Design Center
Pune 411013, India
\{smita.ghaisas, preethu.rose\}@tcs.com

\textsuperscript{b}University of Twente
Enschede, the Netherlands
\{m.daneva, k.sikkel, r.j.wieringa\}@utwente.nl

Abstract—Large IT vendors execute thousands of projects in a variety of business domains and environments. Over the years, they end up repeatedly developing and deploying systems for a given domain in the same country, sometimes even for the same company. It would save them a lot of cost and effort if they could reliably depend on their past experience and draw insights from lessons learnt in the past. However, this requires them to generalize from past projects to a similar current project. In this paper we draw lessons learnt from three industrial case studies on how to generalize by similarity.

Index Terms—Generalizability, Similarity, Industrial case studies, Projects, Researchers, Project Managers.

I. INTRODUCTION

The problem of generalizability from past cases is a common and recurrent problem in industry. It starts right from the stage of bidding for a project. To companies, project organizations and project managers, generalizing means (i) comparing similarities between past cases and present ones and (ii) judging the extent to which similarities and dissimilarities between past and current projects justify or refute the claim that a mechanism observed in a past project, will be repeated in a current project and hence that adopting a similar set of practices would lead to a successful execution of a project. A major threat to validity of these generalizations is that they are made because of subjective and unjustifiable judgments. This can lead to a suboptimal decision on whether or not to adopt “similar” practices from the past. Poor comparison by analogy might well lead to premature adoption of practices only to discover later that they have little fit to the present project context or it may lead to ignoring practices for achieving project results that matter to business.

The aim of our paper is to share what we learnt in our attempt to reduce this subjectivity in the process of similarity-based generalization by identifying systematic ways to judge similarities. Specifically, we attempt to shed some light on two questions (1) what value can a business expect from past academic case study research, when a business faces a new project that appears to be similar to the previously published cases; and (2) How can someone (e.g. a researcher or a project manager) generalize from observed cases by using his/her knowledge about similarities and dissimilarities between past and future cases (i.e. projects) ? With this purpose in mind, we studied three large-scale outsourced projects in a large IT vendor’s organization. The projects were known to us in sufficient detail because we used them between 2010 and 2012 to carry out qualitative case study research on agile requirements prioritization [12]. In the present paper, we reflect on our experiences in these projects and on the reasoning about generalizability of our findings from one context to another. While comparing our project’s contextual settings and how the execution of agile requirement practices was contingent on them, we derived some lessons learnt that we think can be reused by other IT vendor organizations and researchers who want to generalize from past cases to present or future ones. Although we selected requirements prioritization as the process to study generalizability, we think our lessons about similarity-based reasoning could be applied to “similar” RE process, or even to “similar” Software Engineering (SE) process. As for all similarity-based generalizations, this claim must be substantiated by further research that makes clear what actually constitutes relevant similarity here.

Section II reflects on generalizing by similarity, while Section III introduces the three projects we studied. Section IV presents the findings from our case studies. Section V discusses related work. Section VI lists the main contributions and concludes the paper.

II. HOW TO GENERALIZE BASED ON SIMILARITY

Similarity-based generalization is reasoning by analogy. It is existential, not universal; suppose P is any proposition (a claim about the world) and from the observation that in one or more cases, P was true; the researcher infers then that in some future relevantly similar cases, P will be true. The practitioner wants to draw the very specific inference that, because in similar, past cases, P was true, it may also be true in her current case. These are weak generalizations and every application of such a generalization to a current case requires reasoned judgment. For clarity, we can ‘weak’ a

\footnote{Because we discuss case study research, and also because in empirical SE, the question is how to generalize across cases, the generic term is ‘case’. In this paper, we speak of projects as one specific instance of ‘case’}
generalization of the form for some (few, or many or almost all, or even all, but at least for some) entities, P is true. (We note that a weak generalization is the opposite of ‘for all x, not-P is false’). From case studies, a weak generalization is all we can get as far as generalizations go. This is nevertheless useful for the practitioner, because such a generalization provides a heuristic for assessing what factors to look at in the current situation, and what the possible consequences of actions could be.

Reasoning by analogy infers a property of an unobserved case, that we will call the target of generalization, from observed cases, that we will call the source(s) of the generalization. It requires a so-called prior association, which is a relation observed in the source [1, 5]. The analogic inference then proceeds to note similarities and dissimilarities between source and target, and concludes that because of the similarity between source and target, and despite the dissimilarities, the same association will occur in the target.

For example, suppose a researcher has observed in three agile projects that there was no customer on-site of the project, although in all other respects, the projects followed the guidelines of an agile methodology. In all three projects, a software development company performed the agile project for a small company. Then the prior association is:

\[
\text{The project is agile, performed by a software development company for a small company, and there is no customer representative on-site.}
\]

This relates a project characteristic, two actors playing a role in the project, and an observed practice. Now the researcher encounters a fourth project of which she knows that it is agile, and is performed by a software development company for a small company. Is the researcher justified, based on these similarities, to predict that in this project too, there will not be a customer on-site?

Without further justification, this prediction is a wild guess. It might very well be true, but similarity relation so far has given no reason to make this prediction.

For such a generalization to be valid there must be (i) a reason for the similarity to be sufficient to generalize from source to target, and (ii) a reason for which the dissimilarities are not sufficient to block generalization from source to target. Bartha [5] gives several kinds of reasons that have been used successfully in the history of science and mathematics.

In this paper, we will focus on one kind of reasoning, which we call architectural similarity. To find such a similarity, we must decompose the case into components and relationships, such that the interactions between these components produce the phenomenon we want to generalize. For example, in the agile example we have a simple architecture consisting of two components (vendor and client) and an interaction (the agile project). One of these components, the client, is small. We now postulate a mechanism that could have produced the phenomenon. The mechanism consists of two parts: (1) because the client is small, it has limited resources, and this is the reason that the client does not put a customer on-site. (2) At the same time, if the vendor wants repeat business of the client, they will assign a developer to play the role of ‘on-site developer’ (which is meant to replace the missing on-site customer) and verify her decisions with the client if and when needed. Suppose that re-visiting the case descriptions reveal that this has indeed been the case in all three projects. Then this mechanism explains the prior association. This is a reason to apply the generalization to the target. This is still a hypothesis about the target to be verified; one way to gain trust in this generalization is to check the availability of human resources at the client and the presence of a trusted developer at the vendor site.

Generalizing from this example, the researcher may search for an architecture present in the past cases, where an architecture is a collection of case components and interactions. The components are actors in the cases, and these actors have capabilities, such as being able to pay for an agile project, as well as limitations, such as human resource limitations. These capabilities and limitations create certain mechanisms that have certain effects. By a mechanism we mean an interaction between the case components that produces overall effects. For example, because the client is small (limitation), it has limited human resources (limitation), so it will not include an on-site customer in the project (mechanism). And because the vendor has a developer trusted by the client (capability), it will appoint this developer to act as an on-site customer on behalf of the client (mechanism).

These explanations of the observed phenomenon that there is no on-site customer give reason to predict that in cases with a similar architecture, similar mechanisms with the same effect will occur. This is not a deterministic, fail-safe argument. The researcher has to check whether in the next case with the same architecture, the same mechanisms will indeed occur. Maybe in the fourth case the client is low on business or believes in working according to the book, and does put a customer on-site. However, this argument provides more support for the generalization than the mere claim that similar cases will contain similar phenomena. The two identified mechanisms can be viewed as theories that have validity beyond the three cases in which they were observed although it is not known exactly for which other cases they are true. Any application of a theory must deal with the problem that in a practical case, there may be factors not incorporated in the theory that may cancel the predictions of the theory. In our example, in any particular case there may be mechanisms, capabilities or limitations, not mentioned in the two mechanism descriptions above, that cancel the possible effects of these mechanisms.

This concept of mechanism is familiar from object-oriented programming but it turns out that it is useful to describe and understand social phenomena [2, 3, 4, 8, 9, and 10]. Here, we use it as a basis for similarity-based generalization.

To summarize, our similarity-based inference contains the following steps:

1. Describe the past projects’ architecture in terms of actors and their capabilities.
2. Identify a mechanism created by this architecture and explain this mechanism in terms of that architecture.
3. Assess if a target case will exhibit the same mechanisms, assess similarities and dissimilarities between the architectures of sources and target cases and explain why the similarities are sufficient to justify the generalization and the dissimilarities not sufficient to block the generalization.

This way of reasoning is quite natural for project managers, as the examples in Section IV illustrate.

III. PROJECT DETAILS

In this section we introduce the three projects from which we draw our experiences. The projects were called “agile”, but in reality a mix of agile and structured practices were used. The interaction that it provides to the client incorporates all the agile practices that are required: delivery of working (and tested) software in regular increments, intensive contact with the client, incorporating changes as appropriate, and on-site customer representation throughout the project. However, the project was organized along traditional lines, hierarchical in structure and supported by extensive documentation. It is a core value of the vendor to satisfy the client, and what a mature organization can do is set up processes accordingly. The projects were used in our earlier research [12] on agile requirements prioritization where we compared execution of agile requirements prioritization practices within a large vendor organization. The contextual characteristics of the projects are described in much detail in [12].

All three projects used Agile, a form of Scrum that was adapted to the specific project settings. All the projects practiced Agile RE, with (re-)prioritization occurring regularly. All were distributed projects, with at least two geographically separated locations being involved. For confidentiality reasons, we refer to these projects as project Alpha, project Beta, and project Gamma. The three projects had a number of broad similarities, as well as differences. All projects were outsourcing contracts. In terms of scope, projects Alpha and Beta were large. Gamma on the other hand, was a medium size project. Concerning the nature of contractual agreement between vendor and client, Alpha was a fixed-price/fixed-duration project, while Beta and Gamma had a flexible timeline.

We make the note all three projects have been considered successfully completed by the client. In what follows, we describe the contextual setting of each case study project in more detail.

- **Project Alpha:** The practitioners in project Alpha were part of two programs involved in the development of a large software solution for automating insurance processes. The practitioners were experienced in working on large projects. The development project teams were co-located, and the client teams were dispersed globally. The case study participants included: one Scrum master, one business analyst, two business analyst leads, one delivery head (responsible for transforming the clients’ user stories into ‘delivery stories’ that include architecture design decisions and non-functional requirements), one portfolio manager (who was responsible for the group of client’s projects managed as a portfolio), and one test scenario team lead (responsible for end-user acceptance testing and making sure requirements are testable and verifiable).

- **Project Beta:** This project’s team had around 35 to 40 people, whose function was to complement an existing development team on the client’s side, leading to an overall project consisting of 150 people. The team was in place for three years to enhance the client’s existing development effort using agile methodology. Various products to be developed, were divided into ‘releases’, with each release consisting of sprints of either two or four weeks. Requirements here were in constant flux, with the mind-set to accommodate requests from the business whenever possible. Requirements changes were taken up in between sprints, not during a sprint.

- **Project Gamma:** In this setting, the client was internal, but not co-located. The project team developed a product for an internal client, who then used the product to develop solutions for an external customer. For the purposes of this paper, we will refer to the system development team as the ‘development team’ and the internal client’s team as ‘the client’ (as there was little or no direct interaction between the first development team and the external client). The project consisted of 85-90 people, with 35-40 people on the developer’s side and 50 on the client’s side (again, we refer here to the internal client). The product was a model-driven development environment. Gamma development team was the actual owner of the product, and used its interaction with the client as a way to expand the product with the aim to make it widely applicable in a broad range of settings. As the client used the tool for their own development, they actively discovered bugs or limitations and sent the change requests to the development team. The development team then tried to accommodate these requests as quickly as possible so as not to slow down the client’s development effort. In contrast to Alpha and Beta, the requirements for the product in Gamma were not only determined through feedback from the client, but were also influenced by the fact that the development team had its own long-term plans for the tool. The development team switched to Agile only four months before the interviews were conducted. Individual sprints were four weeks in duration at the start of the transition, but were later extended to six weeks to allow for more time for testing and consolidation. Between the sprints, a period of one or two weeks was used for planning the next sprint (in the other two cases, this happened concurrently). As with project Beta, changing requirements were accommodated whenever possible, but intra-iteration re-prioritization was kept to a minimum.

IV. FINDINGS

This section presents the similarities we observed in the three projects and analyzes them to detect underlying mechanisms.
for similarity-based generalizability. Though our findings are in the context of requirements prioritization, the mechanisms can operate to cause similar effects in any large outsourced project.

Observation 1: A ‘delivery story’ artifact that was devised to complement user stories and was adopted in each project, allowed harmonizing the goals of the clients and those of the vendor [12].

Supporting Quotes/Examples:
Project Alpha
“It is not easy to comprehend from the business rules and directly develop. So, we had formulated delivery story idea.”

“ [. . .] delivery story is like a pre-development work for the Scrum”

The project teams (of Alpha, Beta and Gamma) conducted user story workshops and delivery story workshops with customers. This is the particular manner whereby the large vendor team created rapport and built trust with the clients.

Mechanism: Designing an artifact and a work process around it meant specifically to meet user expectations of new delivery methods and techniques balances the challenges involved.

Observation 2: Each project had a client willing to co-own parts of the delivery process

Supporting Quotes/Examples:
Project Beta
“Business analyst will have interaction with the business customer, and the Project Owner. The business customers were on the client side. Business analysts were both on-site and off-shore and the Project Owner was from client side. I’m specifying this from client side to say that they take ownership, but we will be included in all the...”

Agile and Collaboration:
“This Agile made us feel that [client] was part of [vendor organization]. So we did not find any major difference.”

“If it was waterfall, only the project manager interacts with client and gives us the information. But here it is not that”

“Agile is about making the entire team understand what they are going to do and deliver. Including business and IT. Typically, what happens is, the business spends some time, defines the requirements, they forget it and give it to IT. But here, once we make both the parties understand what we are going to deliver in terms of business value, then both will work together and will try to help each other”

“It’s not like ‘we’ and ‘they’ kind of approach: it’s really one single team, which has actually benefitted the project big-time.”

Our case study analysis revealed the importance of process co-ownership by the client. We observed the client organization proactively took ownership by playing the roles of Product Owner, Project Owner and Subject Matter Experts in the project development process. The decisions for sign-off is made by the Product Owner and Project Owner is responsible for overseeing the project and for making sure that the right changes are communicated to the vendor in a timely manner (e.g. before the vendor’s team starts implementing the requirements that would be subjected to changes). The product owner is advised regarding business decisions by Subject Matter Experts who are also representatives of the client’s organization. We found the choices for these roles could influence co-ownership to a great extent. As indicated by the participants, Agile decreases the gap between the business and the development team, and actually helped promote the idea of everyone being part of a single team.

Mechanism: Forming a co-ownership with client for new experiments in large complex projects reduces the risk of failure.

Observation 3: In each project the client and the vendor had complementary specialized knowledge that helped in aligning technology with business.

Supporting Quotes/Examples:
Vendors’ Domain knowledge
“We have a person specifically responsible for the domain knowledge in the team-business analyst, they understand the domain. So there are business architects, technology architect. Business architect works closely with the domain people. And then we have enterprise architect, they understand both.”

Clients’ Domain Knowledge:
“When we didn’t know something we went and asked the domain owner from client side and it was just on the spot we clarified and they provided that knowledge.”

Mechanism: If knowledge and skills of client and vendor complement each other, chances of business and technology alignment are enhanced.

Observation 4: In each project the client had a buy-in for the change management process.

Supporting Quotes/Examples:
Project Alpha:
“It’s a combined decision taken by the client and vendor team representatives taking into consideration the impact, estimates etc. We had a formal change management process”

Mechanism: If the client is actively and continually involved in decision making, she is more likely to support the outcome of the decision.
Observation 5: Each project had a client with an explicit preference for method, technology, architecture, tools that the vendors did not necessarily have expertise for.

Supporting Quotes/Examples:
Insistence on using Agile (Client):
“The desire to use agile development methodologies in our cases came from the client, rather than from the developer’s organization. “

Participant from project Beta stated specifically that the reason for using agile is that, for the client, previous projects were not able to reach the market on time and did not meet the customer’s expectations. The primary reason why the client organization opted for agile was to minimize turn-around time. In project Alpha, usage of agile was also a request from the client’s side for similar reasons. There were some change management issues when agile was first introduced.

Project Beta: “In terms of Agile, the clients themselves were adopting Agile, and there was quite a lot of opposition to change, from Waterfall as well as a lot of standard process in terms of documentation and all that. So it was a slow process. You have to change the people’s mind-set to adopt Agile, once you are into Agile.”

Participants from all projects mention a learning process when agile development was first adopted; it took a while for everyone to get used to working in iterations. There were some initial hick-ups (mainly planning issues) due to inexperience with agile, but these cleared up after one or two iterations. According to a participant from project Beta, it took around three or four months for the project to get fully used to the Agile methodology.

Mechanism: If the clients insist that vendor must work on a given method, technology, architecture, tools, learning curve and delays thereof are to be expected.

Observation 6: Each project organization had a designated role responsible for effective communication.

Supporting Quotes/Examples:
Participants indicated that communication was facilitated in various ways; just using email was found to be insufficient and caused a significant lag in communications, sometimes delaying efforts by a day or more. Interaction was augmented by more direct means of communication, starting with instant messaging. An application called Webex is used for showing screenshots and giving demos while another application, the instant messaging tool was used to facilitate direct video-chat between people from the client and the vendor. The client played an active part in setting up this arrangement, installing webcams and enabling direct lines of communication. The client also helped facilitate distribution by extending support after office hours to increase overlap between different time zones.

Project Beta: “The coordinator is responsible to make sure that the communication happens properly and that whatever requirements have been communicated appropriately to their off-shore counterparts”

Project Gamma: “Every day there is a 15-minute Scrum at ‘off-shore’. Between on-site and off -shore, the same happens between the coordinators. Later an intermediate Scrum will take place between the top persons from each team. There is reporting to the higher-level analysts on a weekly basis. Also, at the start of iteration, everyone gets together for planning. A similar meeting is held at the end of each iteration for a retrospective”

The quotes indicate that each project had an effective communication mechanism and clients actively participated in all the communications. The quote from project Beta highlights the important role played by communication coordinator. The case study projects had coordinators in place on each of the separate locations to ensure that interaction went smoothly.

Mechanism: An effective team communication mechanism is in place where there is a role responsible for communication within teams and between the vendor and client, so that good working relationship is ensured at both levels.

Observation 7: The understanding of ‘risk’ by client and vendor was project context specific.

Supporting Quotes/Examples:
Participants emphasized that risks as perceived by clients and risks as perceived by vendors are different. They pointed out that in a fixed price/fixed schedule project, additional effort (e.g. adding new project staff) pertains solely to the vendor. But the vendor would then have to ‘spend’ (elsewhere) employable people at no extra cost. The vendor would try to minimize this, if an effort-intensive user story is deemed high-priority. They explained that if a project is not fixed-price, it becomes a risk for the client and there would be intense deliberations as to whether the effort being proposed by the vendor is really justified.

Mechanism: A fixed price contract creates risk for vendor whereas time-and-money contract creates a risk for the client while experimenting with new methods and techniques.

V. RELATED WORK
We looked into published ESE and IS literature discussing the concept of generalizability in case study research. First, we
note that we could not find many publications on this topic and what we found [6, 7] addresses the topic from researchers’ perspective (i.e. the theoretical standpoint is over-represented while practitioner’s standpoint is almost ignored). In [7], the authors emphasize the importance of prior knowledge when one generalizes from a current (large or small) sample to future ones. We agree with those authors and in this paper, we indicate some specific instances of ‘prior knowledge’ that are so useful: (i) knowledge of ‘project architectures’ and (ii) knowledge of mechanisms. We note that the importance of mechanisms has been put forward in methodological literature in social sciences [8, 9, 10] and this strengthens our belief that mechanisms might be a powerful however under-utilized instrument in studies in ESE. We therefore feel motivated to do further research into using mechanisms for the purpose of generalizability evaluation across cases and also invite other researchers who work with companies on industry-relevant studies to join us in this discussion.

VI. CONCLUSION

This paper makes two contributions: First we reflected on what value a business organization can expect from hosting case studies for a researcher. We found that the empirical material collected adds value in at least three ways: (i) it provides rich case descriptions which can serve generalization purposes should the company face a similar project context in which it is supposed to execute a new project with predictable chance of success; (ii) the cases allow for distilling a number of mechanisms that explain why past projects were successful in implementing certain software engineering practices; (iii) using knowledge about these mechanisms and knowledge about the new project, business decision makers can do their own evaluation about the extent to which it is realistic to expect success from implanting practices from past projects in the new context.

Second, we made a proposal for a systematic procedure for generalizing based on knowledge about context similarities pertaining to new and past projects. We demonstrated the procedure on three case studies in outsourcing context. We presented what lessons can be learnt through this exercise.

It is our expectation that these two messages can contribute to a broader conversation between practitioners and researchers on generalizability from case studies and how it benefits businesses around the world. We therefore invite other interested members of the ICSE community to share their own case study experiences and help increase the learning on this important topic.

ACKNOWLEDGMENTS

We would like to thank the TCS Research Sabbatical program for supporting this research.

REFERENCES