The Journal of Systems and Software 157 (2019) 110393

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Exploring the gap between the student expectations and the reality of teamwork in undergraduate software engineering group projects

Claudia Iacob[a,*], Shamal Faily[b]

[a] Buckingham Building, Lion Terrace, Portsmouth PO1 3HE UK University of Portsmouth, Portsmouth United Kingdom
[b] Bournemouth University, Bournemouth, United Kingdom

ARTICLE INFO

Article history:
Received 5 January 2019
Revised 8 August 2019
Accepted 12 August 2019
Available online 12 August 2019

Keywords:
Software engineering
Project
Group project
Engineering education

ABSTRACT

Software engineering group projects aim to provide a nurturing environment for learning about teamwork in software engineering. Since social and teamwork issues have been consistently identified as serious problems in such projects, we aim to better understand the breakdown between the expectations teams have at the start of a group project and their experiences at the end of the project. In this paper, we investigate how 35 teams of undergraduate students approach software engineering group project courses, and how their previous experience with collaborative software development matches their expectations for group work. We then analyse the retrospective documents delivered by the same teams at the end of a 27-week software engineering group project course, mirroring the expectations at the start of the project with the realities described by the end of it.

1. Introduction

Software Engineering group projects entail students working as a team to build a software system. Typically, this might be a product for a client, but the teams might also work on a product or Minimum Viable Product they wish to bring to market. Group projects provide invaluable exposure to the teamwork and project management challenges they will face as practitioners. However, such projects are delivered late in students' education programme (eg. second year at university), or in parallel with several other courses. The structure of such projects mostly focuses on the delivery of the software system, and may not always give teams the chance to reflect and learn from their experiences.

Introducing teamwork in undergraduate software engineering projects can be a double edged sword. Teaching software engineering without group work fails to achieve a fundamental teaching objective of any software engineering course - exposing students to the technical and non-technical work needed to develop software as a team. However, due to the challenges assessing group work, students can be skeptical about embarking on such a task, and critical of their experience with teamwork; few undergraduates have experience of collaborative software development, and a student project does not provide the same level of career recognition as a "real-world" development project.

Teamwork management and group work typically cause serious problems that affect student experience (Vanhanen et al., 2018), particularly as student non-engagement is more detrimental to other students than it is for typical taught courses (Wiggberg, 2010). To help students prepare for software engineering group work, it's important to better understand the breakdown between expectations before a project and their experiences afterwards; this has the potential to improve the group project experience for both students and instructors. To this end, this paper investigates student experience and expectations with respect to teamwork by analysing retrospective accounts teams make as a result of working on a collaborative software development project for an academic year. These retrospectives are discussed in the context of an analysis of the same teams' expectations for the group project at the beginning of the academic year.

We consider related work in Section 2, before discussing the background for the group project course we evaluated in Section 3. In Section 4 we describe the research methodology we employed, before detailing the results in Section 5. As part of Section 5, we identify the expectations students have in terms of teamwork at the beginning of the year, and mirror these expectations to the reality they describe when discussing a retrospective account of the group project. We conclude by proposing strategies to narrow or bridge the gap between students' expectations and the reality they describe in Section 6.

* Corresponding author at: Buckingham Building, Lion Terrace, Portsmouth PO1 3HE UK University of Portsmouth, Portsmouth, United Kingdom.
E-mail addresses: iacob@port.ac.uk (C. Iacob), sfaily@bournemouth.ac.uk (S. Faily).

https://doi.org/10.1016/j.jss.2019.110393
0164-1212/

2. Related work

2.1. Collaborative learning and group projects

Group projects are a form of collaborative learning, where two or more people learn or attempt to learn something together (Dillenbourg, 1999). They have been used in a range of contexts in Computing courses, ranging from training in collaborative software tools (Steeples et al., 1996) to showing groups how persuasion might be incorporated in interaction design (Ashaikh et al., 2016). Group projects also provide students with a visceral experience of the patterns of group-work associated with successful and unsuccessful projects (Oliveira et al., 2011).

As the speed of software development has increased, group project teaching has similarly evolved to meet it. For example, "hackathons" have been used to demonstrate how groups can quickly prototype IoT systems (Richard et al., 2015). However, while the experience of building a complete, tangible system is important, students also need time to develop their ability to reflect, self-assess and self-evaluate (Daniels et al., 2011). As Software Engineering is concerned with the application of engineering principles to the production of software, software engineering projects give students the time necessary to develop this relevant expertise. Moreover, Software Engineering group projects are flexible enough to expose students to a wider range of challenges, such as learning how software is developed across organisational and cultural boundaries (Clear et al., 2015).

2.2. Delivering group projects

The problems faced by comparatively senior students (eg. final year students) completing Software Engineering group projects has been the subject of much work. Dugan (2011)'s review of approximately 200 computer science group courses identified some of the challenges, such as the difficulty students have valuing group work over individual work, and difficulties addressing role allocation. More recent reviews of group projects such as (Bastarrica et al., 2017; Bruegge et al., 2015), and (Molléri et al., 2018) report similar experiences. However, such reviews provide an instructor perspective of group projects; they do not take into account the student perspective of courses.

Briggs (1991) discusses the delivery of a second year Software Engineering group project, and reinforces the difficulty that students experience both engineering software and working in groups. This course is, however, atypical from more recent software engineering group projects in two ways. First, the course's primary objective is imparting the experience of software engineering group work; the teaching of software engineering techniques is a secondary objective. As such, although groups produce deliverables, group members are assessed solely on a personal review of their particular contribution. Second, the burden of learning and delivery is comparatively light. The course described ran for a nine-week period, and was explicitly timetabled during teaching periods conducive for students completing the project, and instructors who are given ample time for marking.

Briggs summarises the difficulties faced by second year students without detailing specific challenges faced. Schilling and Sebern (2013) describe some of these challenges while considering how a senior software engineering course was adapted for a sophomore audience (i.e. second year undergraduate level) over multiple course deliveries. They found that minor gaps in knowledge can be overwhelming to sophomore students given the other material they are also expected to pick up over course delivery. While the timetable issues reported are typically a problem found in group projects undertaken by more senior students, Schilling & Sebern claim that – as a result – sophomore students may not have the ability to accurately assess their own learning needs, but the basis of this claim is not made clear.

Assessing learning needs might be carried out using retrospectives. Their use in software engineering project courses was recently considered by Sedelmaier and Landes (2017), who described several approaches for teaching soft skills relevant to more senior software engineering students. Students were free to choose their own proposed system, team mates, and process models. They found that students found it difficult to juggle technology and project management. However, they also found that the self-reflection resulting from completing the retrospectives helped students better understand how well they were achieving the course's learning outcomes. A critical analysis on the impact of retrospectives on subsequent performance is outside the scope of our work. We note, however, that examining the relationship between retrospectives and future performance would be an interesting avenue for further investigation, particularly if differences were noted between second and final year students.

2.3. Student perceptions of undergraduate group projects

Doctoral research by Wiggberg (2010) contrasted expectations that instructors have around group projects with student expectations. His work found that assessing a produced artifact leads to less collaboration, increased stress and fewer learning opportunities, and perceived stress can hold back learning. He suggests making students more aware of the desired learning and learning outcomes at the outset of any project.

Peters et al. (2015) examined the perception of students by analysing the student evaluations both before and after the completion of an open ended group project unit (Daniels, 2011). They found that the root cause of frustrations experienced by students was a lack of appreciation of the role they, as individuals, were making to the broader software engineering context. As such, while students appreciated the opportunity to improve their professional skills, Peters et al. (2015) propose that group projects should be framed as an opportunity for students – as problem solvers – to develop their problem solving experience using complex, open-ended problems. These results are consistent with a more recent qualitative study by Isomöttönen et al. (2019) on the student perceptions of group projects as enabled learning environments. They highlight that problems experienced by students alternating attention between group projects and other courses should be framed as an opportunity for complex learning in a professional work environment, and that students should be made aware of this and the importance of student commitment before commencing the group project.

Raibulet and Fontana (2018) report on the experiences of student project teams using software tools to foster team collaboration. Although the feedback indicated that many students found tools like GitHub and Microsoft Project useful for managing tasks and communicating with team members, there is no explanation about why this might be the case. The feedback also indicated that student teams treat project management and software quality only as a secondary concerns, and despite the apparent utility of the collaboration tools teams largely relied on informal meetings and chat applications for collaboration.

Holmes et al. (2018) surveyed students both before, during, and after participating on a group project to understand the perceptions held by students. Students appreciated the teamwork and collaboration opportunities afforded by projects compared to other courses, and the particular role of collaborative processes and tools. They also found the novelty of the tasks they were undertaking make time management and estimation difficult. However, while the authors collected feedback before and after the project, the results are predominantly based on feedback obtained post-

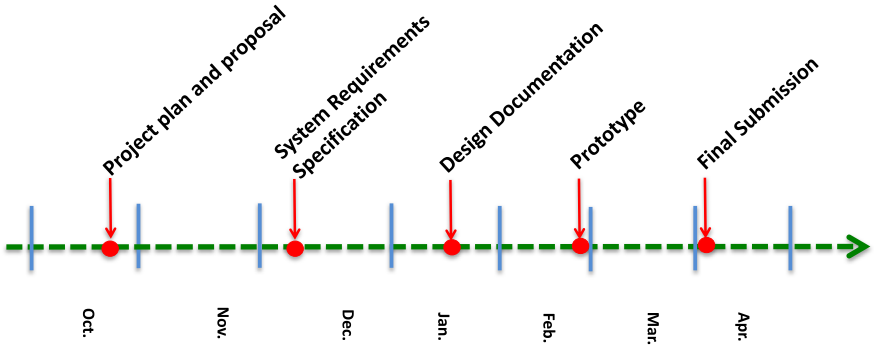

Fig. 1. Group project timeline.

project, so the results do not consider the evolution of student perceptions.

Delgado et al. (2017) consider student perceptions of agile practices when describing the evolution of a project-based course over several semesters. While development-centered practices like simple design, continuous delivery and continuous integration were popular and commonly adopted, students reported problems associated with task estimation, work distribution, and intra-team communication.

3. Group project background

As part of an annual second year undergraduate course, we ask students to work in teams of 5–6 students to design and develop a medium size software system of their choice. We define medium complexity as a system including an element of storage, one or more software component encapsulating some application logic, and a graphical user interface component. The course is run for 27 weeks, from September to March, and students are asked to submit deliverables throughout the year. The deliverables include: 1) *Project plan and proposal (Plan)* worth 10% of the final mark for the project, 2) *System requirements specification document (SRS)* worth 25% of the final mark for the project, 3) *Design documentation (Design)* worth 25% of the final mark for the project, 4) *Prototype (Proto)* which includes a demonstration of the prototype, the source code, and the test cases written for the system. The Prototype is worth 30% of the final mark for the project, 5) *Retrospective account of the project (Final)* worth 10% of the final mark for the project.

The deliverable deadlines are spread out throughout the 27 weeks as depicted in Fig. 1. Teams are given 3 weeks to write a project proposal and plan for a software solution of a medium complexity problem of their choice. Once the project proposal is approved by the course coordinator, teams have 4 weeks to gather, specify, and validate an initial set of requirements for the system they are developing. Because they do not have allocated clients, teams are required to conduct their own research to elicit their software requirements. The results of this process are submitted as the requirements specification document.

As part of the design documentation, teams are required to provide models of the software solution under development that reflect its architectural design, data and interface structure, and additional logic layer modelling such as components interaction or behavioural model. We mandate this to help students appreciate how these documentation artifacts are linked, together with their role in supporting and communicating their thought processes (Clear, 2003). The writing process of each of the documents also helps students practice diverse and complementary software engineering skills - interviewing end users, analysing requirements, abstracting software solutions, and drafting a plan.

Irrespective of the software development process put in place, teams are expected to submit a video demonstration of the current state of they systems they develop by the end of February. This demonstration should not be longer than 5 min, and it should showcase all the features developed for the system and how they work. The source code and the test cases written for the system must be under version control and access to these repositories should be provided by all teams. The final 4 weeks of the year are dedicated to writing a 3-page retrospective account of the project, where teams are required to answer the following questions:

1) What was the software development process followed?
2) How did the process followed match the initial plan set out for the project?
3) What were the main challenges faced as a team, and how were these overcome?
4) What would you do differently if you had to carry out the project again?

Teams are encouraged to write these retrospective documents to accurately reflect the challenges and the struggles they faced throughout the development process. The questions the retrospectives need to answer provide a common template for all documents. However, teams are encouraged to provide truthful representations of their own experience with the project. While all teams are free to put in place any process for writing these documents, we recommend they have an initial team discussion around the four questions before deciding on the process to put in place to answer them. As all team members need to contribute to the writing of the retrospective account, this initial discussion is designed to help the team get on the same page with respect to the overall team experience with the project.

All the deliverables are group submissions. In terms of assessment, the five deliverables are categorised as: documentation (i.e. the project plan and proposal, the requirements specification document, the design documentation, and the retrospective account), code (source code and tests), or demonstration. For all the documentation deliverables, students have the option of identifying individual contributions to the work, either by naming the authors of each section in the deliverable, or evaluating individual contributions in terms of percentages. The mark assigned to the deliverable is assigned to all team members who have been identified as contributors. The code submission is marked based on individual commits as part of the version control system used. When students adopt pair programming as a technique, they are instructed to commit the code written before switching roles during the coding process. The demonstration mark is assigned based on both the individual code commits authored by each team members and the individually identified contributions to the creation of the demonstration. Team members who have either only contributed code or

only create the demonstration will be awarded half of the marks assigned to the demonstration.

Methods, tools, and techniques for tackling common software development problems are introduced during the lectures. However, students are free to select which ones to apply in their work on the project. For example, in terms of version control management, the lecture describes and demonstrates GitHub. However, in recent years, some teams have chosen version control systems, e.g. BitBucket. Similarly, while the lectures describe interviews, focus groups, ethnography as methods for requirements elicitation, teams are free to choose the method(s) they want to use for eliciting the requirements for the systems they are developing. Teams are provided with two constraints. First, the problem chosen needs to be of medium complexity and approved by the course coordinator. Second, each team is expected to submit the five deliverables throughout the year.

The number of teams enrolled on the course varied from year to year, with 32 teams taking the course in 2016/2017, and 35 teams taking the course in 2017/2018. The software systems developed varied, but all were designed around storage components, a set of minimum five features, and graphical user interfaces. Some examples of systems develop include educational quizzes, product reviewing systems, and finance management systems. Real-world examples are provided for each deliverable; these expose students to the structure and content of such documents. Although their complexity is high, the examples are intended to help students develop the analytical skills required for critically evaluating software documentation.

4. Research method

4.1. Research questions

Our research goal is to better understand the gap between the expectations students have at the start of a software development group project in terms of the teamwork dynamics and the way they perceive the reality of the process at the end of the project. We ask the following research questions:

- RQ1: What expectations do students have in terms of teamwork at the start of a software development group project?
  - RQ1.1: How do undergraduate students perceive their past experience with teamwork?
  - RQ1.2: How do undergraduate students perceive the likelihood of recurring breakdown scenarios reported for group projects to occur as part of a group project they are part of?
  - RQ1.3: How is the students experience with collaborative software development linked to how they perceive the likelihood of recurring breakdown scenarios reported for group projects to occur in a project they are part of?
- RQ2: How do undergraduate students perceive the reality of teamwork after completing a software development group project?
  - RQ2.1: What are the most recurring issues students face during a software development group project?
- RQ3: How do the expectations match the reality students face when developing software as a team?

4.2. Research background

At the end of each year, we collect anonymous student feedback for this course. This allows students to comment on the elements of the course that they appreciated and those they found challenging or thought less of. Among the questions students answer is Q1: "What is the area of this course that needs improving?". For the 2016–2017 academic year, we collected answers from 44 (23%) of the students enrolled on the course. The qualitative feedback collected from Q1 was coded by the first author with each comment being summarised in a few words (i.e. a code) to reflect the core message of the comment. For example, "A lot of people are being held back due to group members not participating" was associated with "lack of engagement from team members".

We extracted all the responses commenting on the teamwork element of the course (representing 70% of all the available comments) and the codes assigned to them, and looked at recurring codes pointing to recurring comments students provided. We define a recurring comment as one provided by at least three different students. We identified 9 such codes, namely *the one-man team, project and work management, lack of leadership, no democratic approach, miscommunication within team, poor time management, lack of engagement from team members, personal conflicts*, and *lack of expertise*. For each code, we extracted all the responses associated with the particular code, and summarised them into the following recurring scenarios:

S1: "I felt like I was the only one doing any work" - *the one-man team*. This describes a situation where one of the team members ended up contributing disproportionately more work than all the other members of the team. In extreme cases, the student submitted an entire deliverable under single authorship.

S2: "As a team, we couldn't agree on how to split the work" - *project and work management*. The situation relates to arguments and miscommunication around decisions that involve the division of work. Some teams struggled with identifying and defining development roles during the group project.

S3: "No one wanted to act as a team leader" - *lack of leadership*. This situation identifies cases where none of the team members volunteered to act as a team leader. In some cases no team member felt comfortable with making decision that would involve the entire team, in others no team member was willing to solely take the initiative with respect to part of the process.

S4: "The team leader dictated what we all should do and I didn't agree with his/her decision" - *no democratic approach*. This scenario refers to teams where decisions were taken by the team leader without much deliberation and consensus. Further down the line, this led to frustration on the part of the team with some team members being stuck in roles they did not feel comfortable with.

S5: "Someone in the team misunderstood his/her task and submitted duplicate work at the last minute" - *miscommunication within team*. The scenario describes situations where the lack of continuous communication among team members led to misunderstandings with respect to agreements on work division. In most of these cases, such misunderstandings only became apparent in the hours leading to the deadline.

S6: "We left the work to the last minute and we didn't have any time to check each other's contributions to the deliverable" - *poor time management*. The scenario was associated with situations where the work on a given deliverable was split amongst the team members with the aim of all the parts being assembled in one coherent document. In some situations, due to poor time management, the teams did not have the chance to read the document as a whole and ensure there are no contradictions once all the individual contributions were assembled in one submission.

S7: "We have not heard from a team member for a long time" - *lack of engagement from team members*. This scenario was associated with situations where team members would stop attending meetings and responding to messages without any notice.

S8: "Conflict arose often among members of the team" - *personal conflicts*. The scenario refers to personal conflicts amongst members of teams. Such conflicts included misunderstandings about individual contributions, miscommunication around

milestones set by the team, or disagreements about decisions made with respect to the development process.

S9: "The team leader/a member of the team decided we should use a specific method or language and then s/he stopped attending the team meetings. No one else knew the method or language and it was too late to change it." - *lack of expertise*. This scenario was associated with situations where one team member confidently supported the adoption of a specific technology reassuring the rest of the team that s/he will play a supportive role throughout the development process. Later on during the development process, s/he either withdrew from the course or stopped attending team meetings.

To answer this paper's research questions, we designed and ran two studies during the following academic year (i.e. 2017/2018). At the start of the academic year, we collected data from individual students enrolled on the course aiming to identify their experience and expectation with teamwork. Throughout the academic year, the students were split into 35 teams, and each team was asked to work on the group project described in Section 3. At the end of the same year, we collected data from each team to better understand their perceptions of the group exercise. We describe the methodology for designing and running each study below.

4.3. Study 1 - teamwork expectations

4.3.1. Data collection and data analysis

At the start of the 2017–2018 academic year, we asked each student to anonymously answer a set of questions to identify their teamwork expectations. Students were asked to both rate their experience with teamwork on a scale from 1 (very limited) to 5 (very strong) (Q1) and identify the number of collaborative software development projects they were involved in (Q2). Additionally, they were asked to: a) identify what, in their opinion, are the benefits (Q3) and the challenges (Q4) of developing software as a team, and b) rate on a scale from 1 (not likely at all) to 5 (very likely) the likelihood of the 9 scenarios reported on by students during the previous year (see Section 4.2) happening in their own team (Q5). Out of the 189 students enrolled on the course, 128 (68%) answered the survey.

All qualitative data collected, namely from Q3 and Q4, was coded by the first author in order to identify recurring themes in the responses provided; these codes were reviewed for consistency by the second author. For Q3, students provided a set of descriptions of what they perceived the benefits of teamwork to be. Each such description was associated with a code which summarised its meaning. For example, "We can get a lot more work done as a team" was associated with "getting more work done". Some responses included multiple descriptions, and were associated with multiple codes. For each code, we calculate its recurrence rate as the number of responses that were associated with that particular code.

Similarly, for Q4, students provided a set of descriptions of what they perceived the challenges of teamwork to be. Each such description was associated with a code which captured its meaning. For example, "It is difficult to find time to meet when everybody has different timetables" was associated with "scheduling meetings". Some responses included multiple descriptions, and were associated with multiple codes. For each code, we calculate its recurrence rate as the number of responses that were associated with that particular code. All quantitative data, namely from Q1, Q2, and Q5, was analysed using IBM SPSS.

4.3.2. Limitations

We asked students to rate a limited number of scenarios, and we did not ask them to foresee any other additional breakdown scenarios. We did not follow up the survey with interviews, so we did not get the chance to clarify the quantitative data gathered. Additionally, this was the first course offered by the programme where this group of students were asked to collaboratively develop a software system. As such, some of the scenarios students had to rate might have been perceived as accidental occurrences, or possibly explained by factors that were independent to the previous year's course delivery.

4.4. Study 2 - teamwork reality

4.4.1. Data collection and data analysis

At the end of the same 2017–2018 academic year, the 35 teams of 5–6 students enrolled on the course were asked to submit a 3-page retrospective account of the process they followed as a team. As part of this retrospective, the teams discussed four topics: details of the software development process they put in place (T1), the way their initial plan matched this process (T2), the issues they encountered and how they overcame them (T3), and the lessons learned throughout the design and development process (T4). All the retrospectives were submitted as team deliverables, and they were anonymised. All teams used the same structure to write the retrospectives, dedicating a section to each of the four topics above (T1–T4).

We carried out thematic analysis of the 35 retrospective documents based on *Straussian* Grounded Theory (Stol et al., 2016) in three stages. First, we tokenised these submissions into small snippets of text, i.e. quotations, ensuring that each quotation reports on a single issue. These quotations were often restricted to one or two sentences, for example *"The beginning of the project started off exceptionally. The whole team met and planned how they would proceed"*. Second, we coded each quotation with a short summary of the issue reported. Such codes were restricted to a few words, for example *"great start of the project"*. One code was often associated with multiple quotations, and each quotation was associated with one code. Finally, we axially and selectively coded these quotations into themes, eg. *"attitude towards the project"*. In summary, 151 codes were used to categorise the 435 quotations elicited, and subsequently grouped into 21 themes. These themes were clustered around the following areas: People, Skills, Process, and Environment.

4.4.2. Limitations

All retrospectives were limited to 3 pages. As a result, teams discussed issues which they felt mostly affected their teamwork possibly ignoring challenges they faced which affected them less or were easily managed. While each team member contributed to the retrospective account, the submission was a group submission. Therefore, we are not analysing individual reflections on the project, but the challenges and lessons learned by the team as a whole. We did not follow up the retrospectives with interviews or focus groups, our analysis being solely based on the written documents.

5. Results

5.1. RQ1 - Teamwork Expectations

In RQ1 we studied what students expect in terms of teamwork at the start of a software development group project. We looked at how students perceive their experience with teamwork in general, and how this matched their specific experience with collaborative software development. Additionally, we explored students expectations for breakdown scenarios in teamwork, and how these expectations matched their own experience with teamwork and collaborative software development.

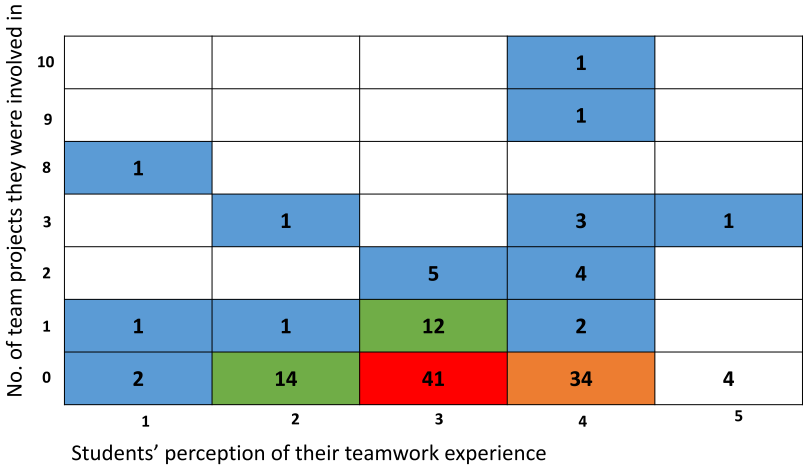

Fig. 2. Each cell represents the no. of students corresponding to the respective a) perceived teamwork experience where 1 and 5 represent very poor and very strong respectively (*X*-axis) and b) no. of team projects they were involved prior to this project course in (*Y*-axis)).

5.1.1. Perceived experience with teamwork

Out of the 189 students enrolled on the course, 128 (68%) answered the survey. 40% of the respondents rated their experience with teamwork either 'very strong' or 'strong', with 15.6% of the respondents considering their experience with teamwork 'poor' or 'very poor'. The mean of the perceived experience with teamwork across responses was 3.24 (st.dev. 0.83). In terms of number of collaborative software development projects involved with, 74.2% of the respondents had not been involved in any such projects, and only 6.3% of the respondents had been involved in more than two such projects. The average number of collaborative software development projects respondents were involved in was 0.59 (st.dev. 1.52).

Out of the 50 respondents who rated their experience with teamwork as either 'strong' or 'very strong', 51% had never been involved in a collaborative software development project. We represent this in the heatmap depicted in (Fig. 2), where the *X*-axis represents on a scale from 1 (very poor) to 5 (very strong) the students' perception of their teamwork experience, and the *Y*-axis indicates the number of team projects students were involved in. The blue and green cells in the heatmap indicate a small number of students reporting to match the corresponding criteria in *X* and *Y*-axis, whereas orange and red indicate a large number of students reporting to match the corresponding *X* and *Y*-axis criteria. For example, 41 students reported being involved in no team projects while at the same time rating their experience with teamwork as 3 on the 1 to 5 scale. Out of those who perceived their experience with teamwork as 'moderate', 59% have never been involved in a collaborative software development project. We conclude that, while students relate to teamwork and recognise it as part of their own previous experiences, they do not necessarily relate teamwork to software development. A lack of experience with developing software collaboratively does not necessarily lead students to reassess their own experience with teamwork. This leads to the false expectation that the skills acquired having worked as part of a team in any other circumstance will easily transfer to the context of software development.

We asked students to identify top benefits and challenges they perceive for developing software as a team. We looked at recurring responses and their corresponding recurrence rates, and identified them in Table 1. While the responses often met benefits and challenges of teamwork, they are neither specific to software development nor concretely pointing to particular scenarios.

Table 1
Student's perception of the benefits and the challenges of teamwork - Codes identified and their respective recurrence rates in percentages.

Benefits	Challenges
Broader perspective and skill set (22%)	Communication (35%)
Getting more work done (22%)	Conflicting opinions (15%)
Delivering work faster (22%)	Unreliable team members (11%)
Sharing the workload (18%)	Scheduling meetings (11%)
Helping each other (15%)	Coordination (7%)

5.1.2. Perceived likelihood of breakdown scenarios

We provided the students with the description of the scenarios identified in Section 4.2 and asked them to rate on a scale from 1 (very unlikely) to 5 (very likely) the likelihood of these scenarios occurring as part of their group project (Figs. 3–7).

S1 ("I felt like I was the only one doing any work") was rated 'very likely' or 'likely' by only 17.2% of the respondents, with the highest number of respondents (38.3%) finding it 'moderately likely'. S2 ("As a team, we couldn't agree on how to split the work") was perceived as 'very unlikely' or 'unlikely' by 46.1% of the respondents, while S3 ("No one wanted to act as a team leader") was found 'very likely' by only 5.5% of the respondents. S4 ("The team leader dictated what we all should do and I didn't agree with his/her decision") was seen as particularly unlikely with only 1.6% of the respondents rating is as 'very likely'. S5 ("Someone in the team misunderstood his/her task and submitted duplicate work at the last minute") was rated as either 'very unlikely' or 'unlikely' by 46.8% of the respondents, while S6 ("We left the work to the last minute and we didn't have any time to check each other's contributions to the deliverable") was rated as 'very likely' by only 3.9% of the respondents. The scenario considered most likely was S7 ("We have not heard from a team member for a long time"), with 6.3% of the respondents rating it 'very likely'. S8 ("Conflict arose often among members of the team") was rated as 'very likely' by the lowest number of respondents, namely 0.8%, while S9 ("The team leader/a member of the team decided we should use a specific method or language and then s/he stopped attending the team meetings. No one else knew the method or language and it was too late to change it") was rated as particularly unlikely with 57.8% of the respondents finding it either 'very unlikely' or 'unlikely'.

Overall, none of the scenarios were seen as particularly likely to occur as part of the group projects students were embarking on,

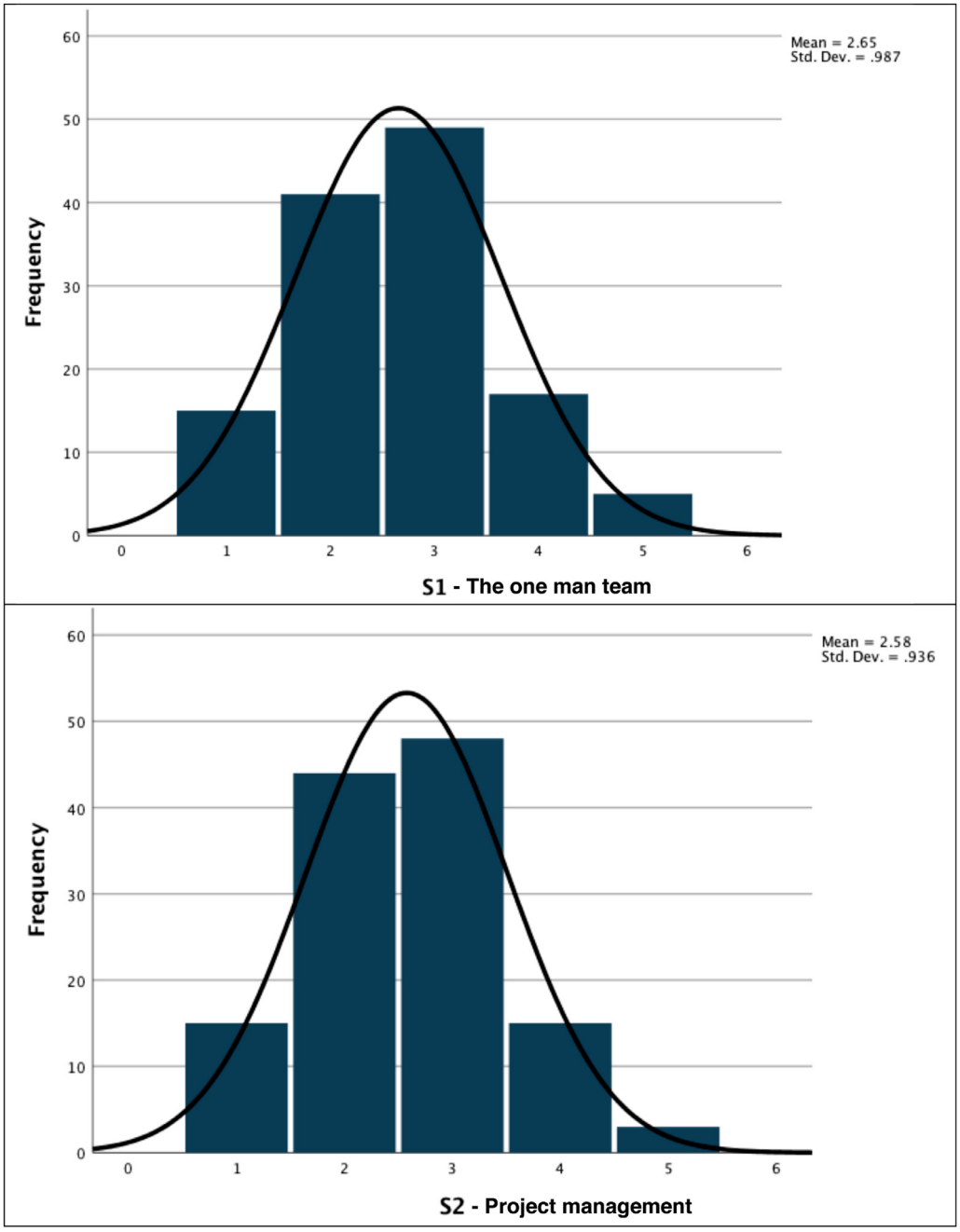

Fig. 3. Students' rate the likelihood of the breakdown scenarios S1-S2 happening as part of their group project: *X*-axis represents the likelihood code (1-very unlikely, 5-very likely), *Y*-axis represents the frequency of code occurrence in responses.

'likely' and 'very likely' having the lowest response rate for all the nine scenarios. Students were not told that these scenarios were the nine most recurring breakdown scenarios reported for the previous year. The fact they did not perceive them as real risks for their own projects reflects the students' expectations and overall optimism they project on a teamwork exercise in software development.

5.1.3. Experience and perceived likelihood of breakdown scenarios

We explored the impact students' experience with teamwork has on how they rate the likelihood of common breakdown scenarios to occur as part of their own group project. The bubble chart in Fig. 8 show the distribution of responses based on the level of experience students claim to have (*Y*-axis) with respect to the likelihood of breakdown scenario (*X*-axis). In comparison, Fig. 9 shows the number of collaborative software development projects students claim they were involved in (*Y*-axis) with respect to the likelihood of breakdown scenario (*X*-axis).

As Table 2 shows, we found no significant correlation between the way students rated the likelihood of the breakdown scenarios and the number of software development projects they claim to have been involved in. We also observed no significant correlation between the way students rated the likelihood of the breakdown scenarios and the way they rate their experience with teamwork. The only exception was Scenario 2 ("As a team, we couldn't agree on how to split the work"), which was moderately negatively correlated with the self-assessed teamwork experience ($r_s = -.311$, $p = .000$).

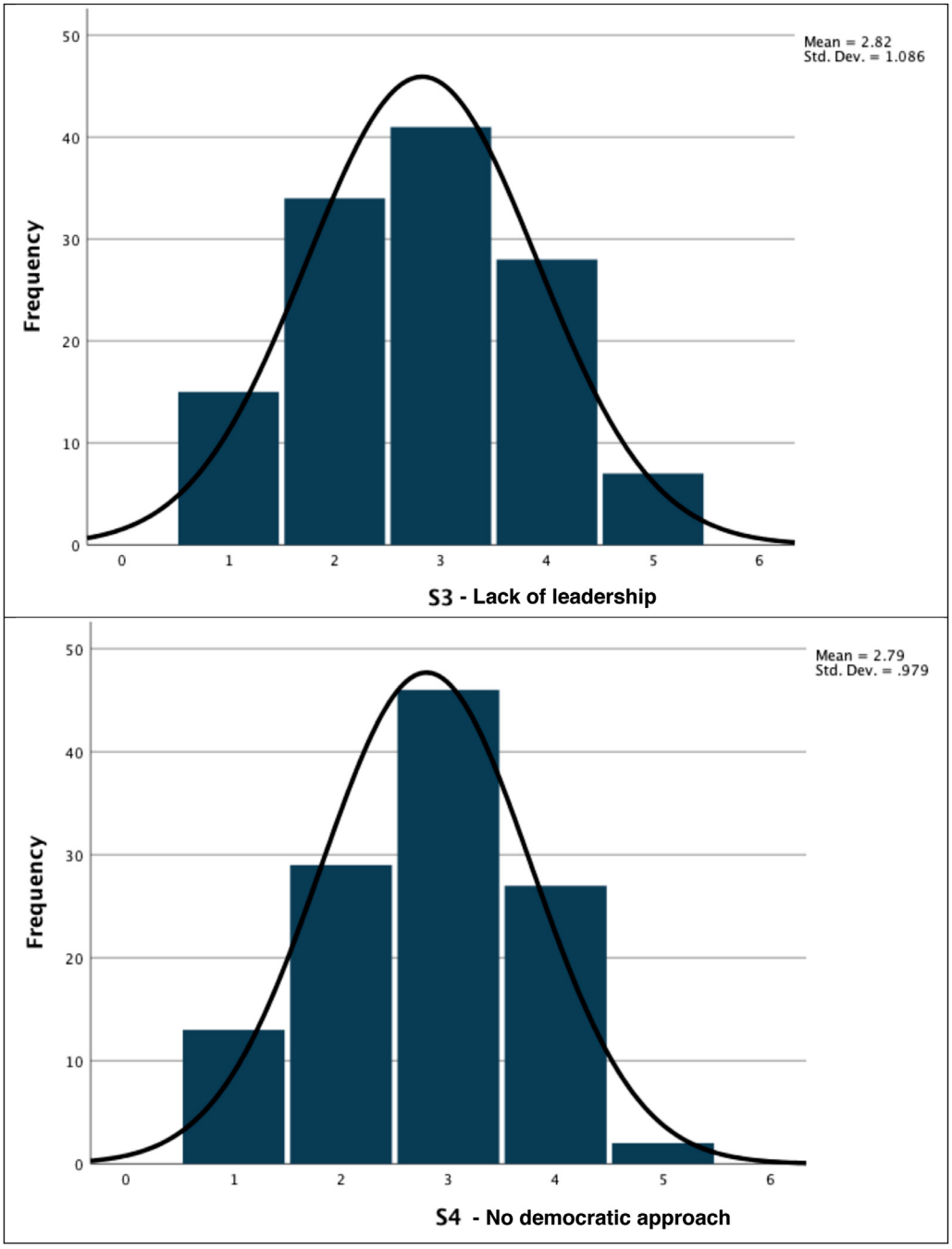

Fig. 4. Students' rate the likelihood of the breakdown scenarios S3-S4 happening as part of their group project: *X*-axis represents the likelihood code (1-very unlikely, 5-very likely), *Y*-axis represents the frequency of code occurrence in responses.

Table 2
Correlation between students' experience claimed/no. of projects students claim they were involved in and the rate they assign for the likelihood of the breakdown scenarios (Spearman coefficient value and *p*-value, respectively)

	S1	S2	S3	S4	S5	S6	S7	S8	S9
Level of experience	−**.178** **.023**	−**.311** **.000**	.087 .166	.022 .407	−**.248** **.004**	−**.215** **.012**	−**.188** **.026**	-.109 .134	−**.173** **.044**
No. of projects involved in	.002 .493	.040 .328	−.030 .369	.069 .230	−.015 .439	-.014 .444	−.144 .069	.144 .123	−.75 .229

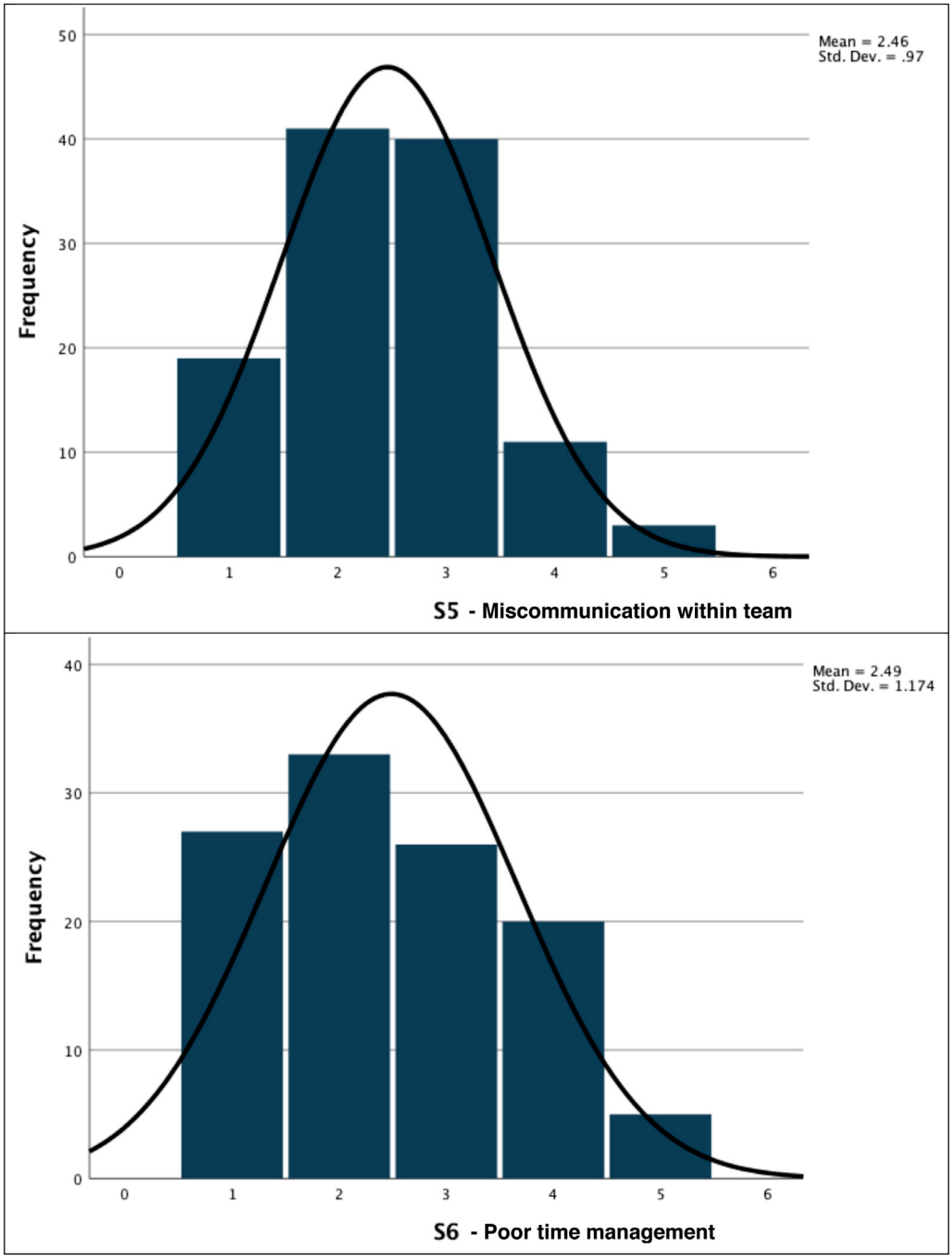

Fig. 5. Students' rate the likelihood of the breakdown scenarios S5-S6 happening as part of their group project: *X*-axis represents the likelihood code (1-very unlikely, 5-very likely), *Y*-axis represents the frequency of code occurrence in responses.

5.2. RQ2 - teamwork reality

In RQ2, we studied how students describe the reality they face throughout the development process of a software system when working as a team. Analysing the 35 retrospective accounts introduced in Section 3, we identified four topics, namely Team (T), Skills (S), Process (Pr), and Environment (E) which cover the 21 recurring themes reported by the teams and depicted in Table 3. The themes in Table 3 are ordered by topic - team (T), skills (S), process (P), and environment (E).

The two most discussed themes across the retrospectives were *team expertise* and *meetings*, with almost 3 quarters of the teams reflecting on the role of meetings in their group project and on how the collective team expertise was identified and exploited throughout the duration of the project. The *development process* and *time management* were themes discussed by more than a half of the retrospectives. While some phases of the development process were dominant (eg. implementation and version control), the teams reflected on other aspects of the development process such as requirements engineering and quality assurance. Other themes discussed by a large number of teams included *team members, role allocation, communication across team*, and *the project plan*. In terms of *team members*, the retrospectives discussed some of the recurring breakdown scenarios related to team membership throughout the project. *Role allocation* discussions included descriptions of strategies followed for assigning roles for the main tasks

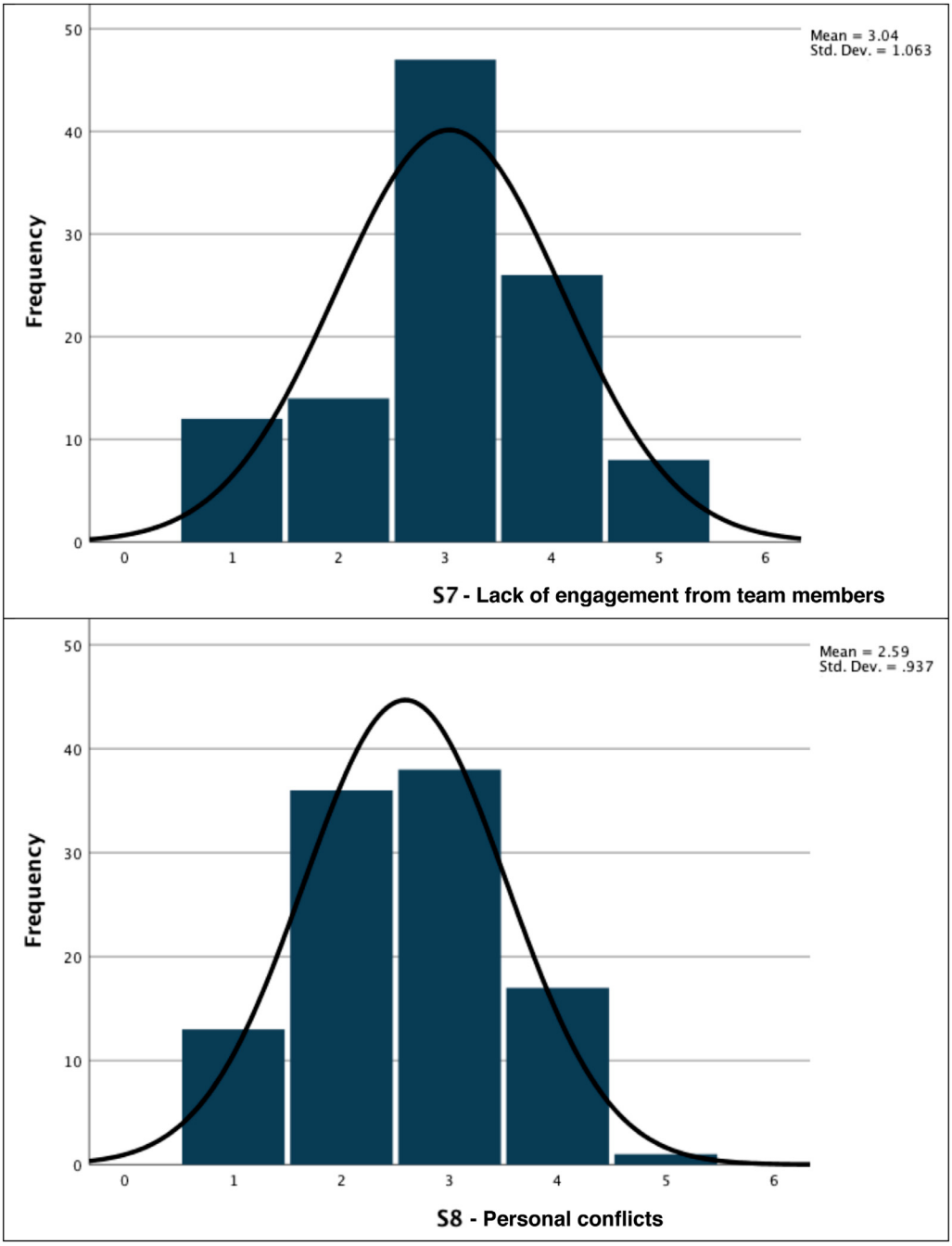

Fig. 6. Students' rate the likelihood of the breakdown scenarios S7-S8 happening as part of their group project: *X*-axis represents the likelihood code (1-very unlikely, 5-very likely), *Y*-axis represents the frequency of code occurrence in responses.

identified for the project. In terms of *communication across team*, the retrospectives identified breakdown scenarios in communication and some of the consequences of these scenarios. Teams discussed their strategies for drafting a *project plan* and following it throughout the duration of the project.

40% of the retrospectives discussed aspects related to *leadership* and the *product delivered*. Discussions around *leadership* revolve around the strategies teams put in place for deciding on team leadership, while all the 14 teams which reflected on the *product delivered* discussed how the software system developed matched the initially planned one. A third of the retrospectives discussed themes such as *coordination* and *team environment*. When discussing *coordination*, teams reflected on breakdown scenarios revolving around the lack of team coordination. In terms of *team environment*, teams reflected on the impact the attitude of the team towards the project and the overall mood of the team impacted the success of the project.

About a quarter of the retrospectives discussed *deadlines* and meeting them, teams' *attitude towards the project*, the choice of *languages and tools used* in the development process, and strategies for *work breakdown*. Less discussed themes included the *shared understanding* across the team and strategies to ensure it, *individual contributions* to the team deliverables and breakdown scenarios around identifying them, and elements of *risk management*. The least discussed themes included strategies for *conflict*

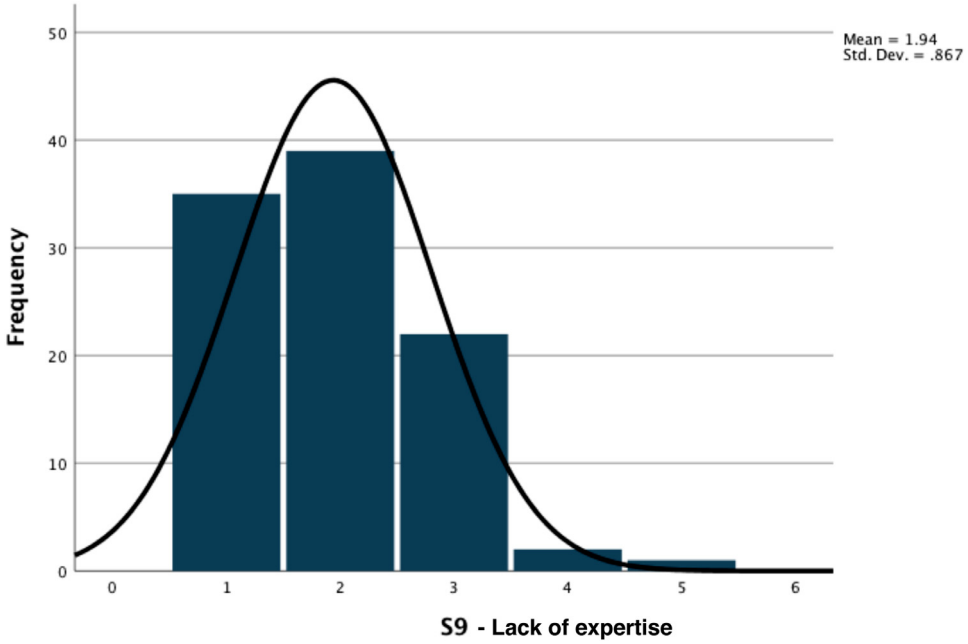

Fig. 7. Students' rate the likelihood of the breakdown scenarios S9 happening as part of their group project: *X*-axis represents the likelihood code (1-very unlikely, 5-very likely), *Y*-axis represents the frequency of code occurrence in responses.

Table 3
Themes discussed by teams for each topic (Team (T), Skills (S), Process (P), and Environment (E)) with the number and percentage of teams reporting on each theme, and the number of sub-themes (codes) associated with each theme.

Topic	Theme	No. of codes	No. of teams	% of teams
T	Team members	5	17	49
	Leadership	6	14	40
	Role allocation	11	16	46
	Individual contributions	2	4	11
	Shared understanding	4	7	20
S	Development process	7	20	57
	Languages and tools used	10	8	23
	Team expertise	14	25	71
P	Risk management	3	4	11
	Meetings	16	24	69
	Meeting deadlines	7	9	26
	Monitoring progress	3	3	9
	Product delivered	1	14	40
	Time management	8	20	57
	Work breakdown	5	8	23
	The plan	5	15	43
E	Attitude towards project	12	8	23
	Communication	11	15	43
	Coordination	7	11	31
	Conflict resolution	3	3	9
	Team spirit	13	11	31

resolution and *progress monitoring*. Below, we discussed each theme in more details.

5.2.1. Team

Team members. A recurring issue across the retrospectives was the lack of commitment of one or more members of the team, with 12 (35%) teams reporting this as an issue. This manifested in poor attendance in team meetings, not delivering work on time, delays in responding to messages sent across the team, or misleading fellow team members. Opportunistic behaviour was also observed by students with team members being absent for long periods of time and only engaging with the project in the days leading to the deadline. The responses to these situations varied. Some team members took it upon themselves to ensure that such situations did not jeopardise the success of the team (*"I have learnt that you can't always rely on other team members to do parts of the coursework on time so I should be more prepared in future team projects to do all I can to ensure I can carry on with my part regardless of what other members have done so far"*). Other teams felt that one member's lack of commitment had ripple effects for the entire team, and decided to react by excluding the team member from the team after repeated warnings. Other teams decided to redistribute the work to the remaining team members and readjust the scope of the entire project to better match the number of students actively involved in the development process. Members leaving the team was another recurring issue, with 9 (26%) of the team reporting that one member of the team withdrew from the project altogether.

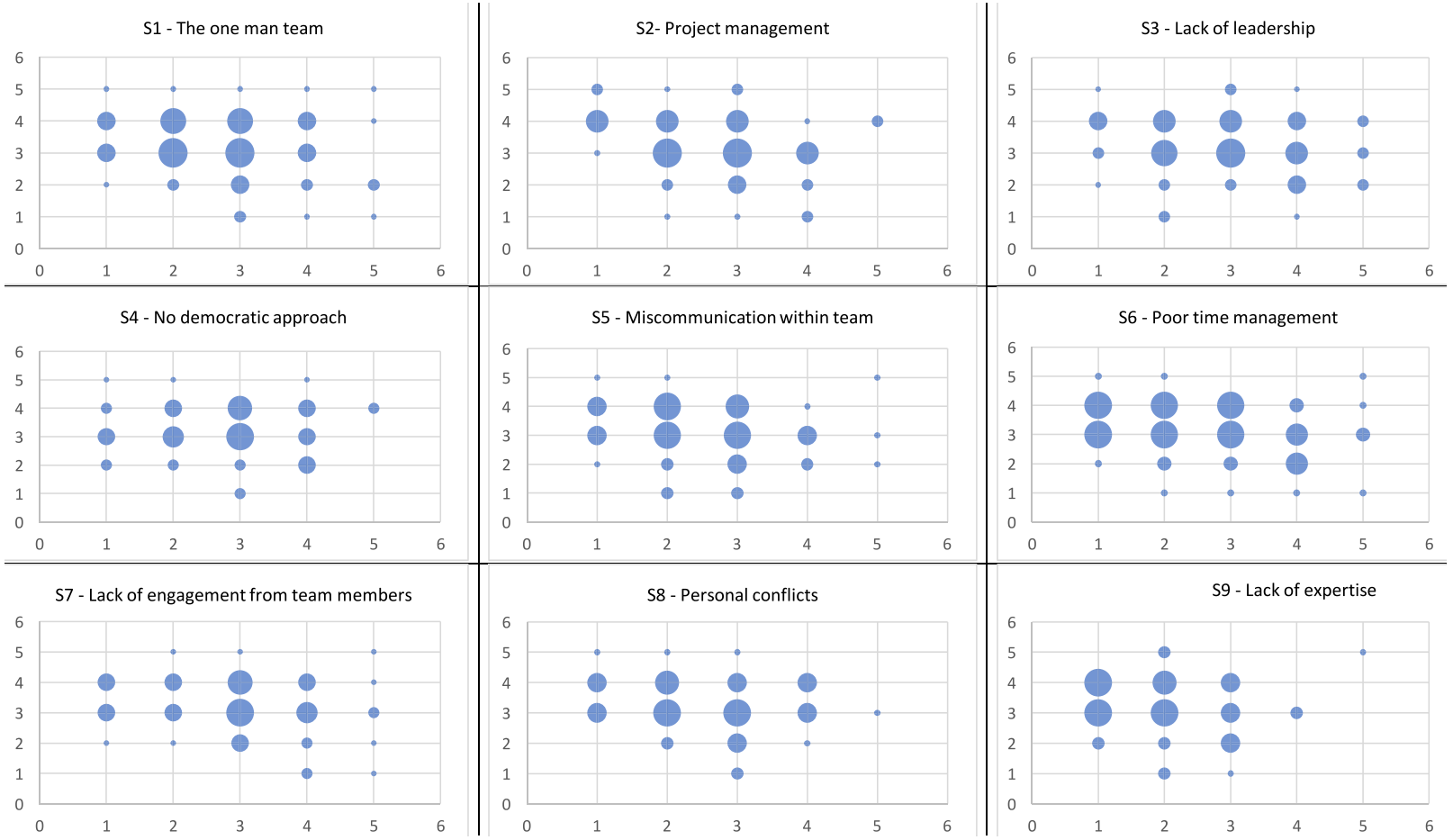

Fig. 8. Distribution of the impact of students' self-assessed experience (*Y*-axis; 1-very poor, 5-very strong) on the perceived likelihood of the breakdown scenarios (*X*-axis; 1-very unlikely, 5-very likely).

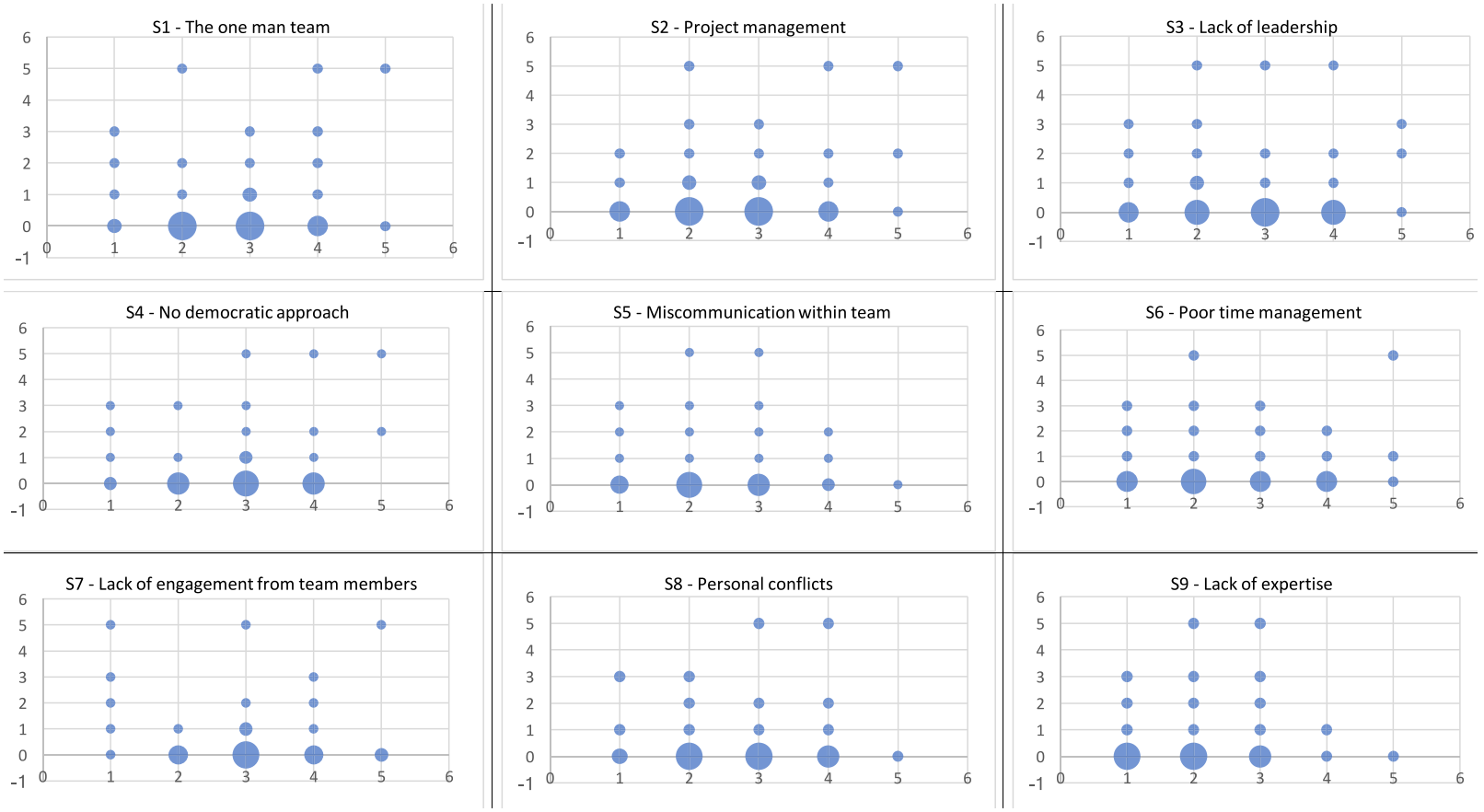

Fig. 9. Distribution of the impact of the number of collaborative software development projects students were involved in (*Y*-axis; value 5 represents 5+) on the perceived likelihood of the breakdown scenarios (*X*-axis; 1-very unlikely, 5-very likely).

Leadership. The attitude towards leadership varied across teams. While 20% of the teams reported a lack of leadership and its consequences (*"This particular project did not have a team leader, but it has been made apparent that there needs to be a figure who is responsible for overseeing the progress of the project and ensuring that tasks are on scheduling and finding solutions if they are not."*), others took an active approach in deciding the leadership roles early in the process. Two main strategies were identified for appointing team leaders. 20% of the teams decided to rotate the leadership role amongst team members so that every member of the team ends up leading a particular development phase (*"At the beginning of the project, we allocated leaders for each submission, this allowed for the coordination and responsibility of each piece of work to be passed from one member to another"*). Taking turns in acting as a team leader has both advantages and disadvantages. On the one hand, it helped all team members build leaderships skills; on the other hand, students felt that some of their peers were more comfortable with the role than others and this had a direct impact on the team as a whole. In 6% of the cases, the team as a whole appointed one team leader for the entire development process, usually as a consequence of a breakdown such as not meeting a deadline, no one showing initiative to start working on a task or the project as a whole (*"Often, when starting a piece of the project, it would take someone to initially start and get the ball rolling before the whole group began contributing."*).

Role allocation. Teams reported several strategies for assigning roles within the team, such strategies being employed either at the start of the project or before the start of each development phase. Some teams used a bidding process where roles were assigned based on individual preference. Others decided to assign roles based on each others expertise, while others split the team into sub-teams and assigned larger tasks to each of the sub-teams. In most of the cases, teams reported that such roles were flexible with students helping each other when in need *"Work allocation was never really dead-set on one person, therefore whoever had finished their part was more than happy to help out other members which had not finished theirs"*. Other teams, however, lacked a clear strategy for assigning roles or did not consider a back up plan if any of the team members did not deliver the work assigned to them. This had a significant impact on the team environment - *"As a team we lacked a clear system of distribution of work and this really took a toll on the less motivated members, which led to a clear difference of involvement and contribution when it came to dealing with the workload"*.

Individual contributions. While the contributions to each of the submissions of the coursework were individual, they were not always even with 10% of the teams reporting opportunistic behaviour on the part of some of their members - *"While some people helped with some parts of the code, others did not write a single line of the software and only showed initiative a couple days before the deadline."*.

Shared understanding across the team. Several teams reported difficulties in reaching a common vision for the project in areas such as requirements elicitation & specification, design, or even implementation - *"I believe however that we as a team, had conflicting visions on how implementation was to be done which can be seen in the SRS and Design Documentation"*. 10% of the teams found it difficult to ensure that all members of the team are on the same page at all times in terms of the progress made and what is left to be done. As response to this, they developed strategies for ensuring shared understanding across the team, including maintaining a shared Q&A document or relying heavily on communication tools such as Slack.

5.2.2. Skills

Development process. The discussion around the development process focused mostly on the software development process put in place (15% of the teams), version control (30% of the teams), implementation (43% of the teams), requirements (15% of the teams), QA (6% of the teams). The lack of experience in using version control systems left a mark as reported by a number of teams and led to time being lost recovering code accidentally deleted or frustration on the part of some team members. Gradually, however, each team chose a strategy to manage such issues, including keeping one branch and increasing communication, assigning different branches to different tasks, or team members and merging these branches collectively or using a simpler tool for version control. Requirements were also discussed either as a constant reminder of what is left to be done, as a way to ensure shared understanding and vision across the team, or to facilitate decision making at later stages during the process. Other scenarios teams reported on included changing the design of the system half way through the process, reusing code to speed up the implementation phase, employing peer-review as a means of checking the final version of each deliverable, and deciding on name conventions to be used across the team.

Language and tools used. With teams being allowed to choose their tools and programming languages, a lot of discussion justifying their choices was inevitable. Overall, we noted two strategies. The first strategy was to give this a lot of thought from the very beginning, and review the languages and tools all team members were familiar with. Some teams took a rigorous approach by asking all members to complete a questionnaire and rate their level of experience with various popular languages and tools. Based on these results, a decision was made to ensure that all team members were comfortable with the choice. The second strategy was to invest a lot less effort at the beginning and choose a popular language, usually one that at least some team members were comfortable with. However, in most such cases, teams ended up switching to another language half way through the process due to unexpected technical difficulties or the realisation that the time and effort required for learning the language to the level needed was unsustainable.

Team expertise. Teams dedicated time at the start of the project to get to know each others technical strengths and weaknesses - *"We started the process by going over our experiences, what we were good at and what we could bring to the project. We looked at some of our past projects and how we could use the skills and experiences we gained in this new project"*. Some used this understanding to correctly estimate their collective technical abilities and, if needed, acknowledge the need for more time and effort to be dedicated to improving their programming skills. Others (17%), however, overestimated the technical abilities of the team and only realised this later in the process, costing them time and marks. In extreme cases, the scope of the project needed to be changed to better fit the skills and expertise of the team - *"We realised that we gave ourselves too many tasks with limited knowledge of how to successfully create them"*. Another issue raised by teams (18% of the teams) was the diverse skillset of their team members. This diversity ensured that even the most complex tasks can be achieved using the collective expertise. This is important given the mastery of complex tasks as a team is an important outcome of a group project unit (Wiggberg, 2010). However, this diversity also led to lack of motivation on the side of the team members less proficient in the various languages and tools used by the team.

5.2.3. Process

The plan. All teams started with an initial project plan. Several scenarios were identified with respect to these plans. Some teams (23%) provided an elaborate plan with little consideration for the resources the team had available in terms of time and expertise - *"Some of the functionality defined in the earlier stages of the project were out of reach and impossible to develop by the time of the submission date"*. Other teams provided a simplistic plan

with little connection to the system to be developed, leading to underestimating the work that needed to be done and leading to rushed work in the days before the deadlined. Finally, a number of 4 teams submitted an initial plan as a requirement for the course, before subsequently ignoring it completely throughout the design & development process - *"The project plan defined in the first coursework deliverable was not referred to by the team while coding the prototype, as team members prioritised a recently created task list in the short-term over the plan"*.

Meetings. Team meetings were a recurring theme across all retrospectives, and played a crucial role in all projects. Teams reported on the regularity of the meetings, with some appreciating the positive role regular meetings played in the team's progress. In extreme cases, little work was done outside of these meetings - *"While team meetings were held frequently, not much work were done afterwards"*. Other teams identified their irregular meetings as the element with the highest negative impact on the progress made by the team. Most teams recognised that logistics matter, so they put in place strategies to ensure the success of their meetings. Such strategies included planning meetings ahead and defining an agenda for every meeting, keeping track of minutes of each meeting, setting a day and a time when to meet every week, or using a register to track attendance.

Meeting deadlines. Meeting deadlines was a strong consideration for all teams. Most teams acknowledged the necessity of putting in place their own milestones, and some teams enforced them for every submission. However, this was not the case for all teams, many of them reporting leaving work to the last minute and struggling to ensure that deadlines were met.

Monitoring progress. Monitoring progress did not feature as a common concern for the teams. While this was unexpected, a couple of exceptions were identified. One team in particular defined and maintained a timetable of the project, allowing the team to *"determine whether we were behind schedule or ahead of it"*. Another team attempted to enforce a progress monitoring mechanism via GitHub, but most team members did not adopt it, leading to situations where *"tasks were being tracked and completed with some members unaware of it, reducing the team productivity and making it difficult to understand where the project development was at in our overall project plan"*.

Product delivered. Fourteen teams (40%) reported delivering a final product significantly different from the one initially planned. The reasons for this varied. Some teams reported difficulties with getting everybody to engage with the project and eventually leading to a reduced team, limited time, and the impossibility of meeting all the requirements initially specified for the system. Other teams blamed an overestimation of the time they had available, and planning features infeasible to implement with the resources available. Still other teams preferred to deliver a working system even if less complex than to risk developing more complex features but not have the time to test them properly.

Time management. The group project was part of six other courses students had to take during their second year. As a result, the main cause reported for poor time management in 32% of the retrospectives was the need to manage multiple commitments at the same time. Some teams felt that this did not give them enough time to learn new technologies or contribute as much as they would have wanted to the work of the team. Some tasks were sacrificed as a result of poor time management and skewed time estimates; these included code documentation, testing, and system evaluation - *"Unfortunately, the latter stages of unit testing, integration and system testing, and operation and maintenance were not reached due to the difficulties of time and amount of work completed described below"*.

Work breakdown. Decisions on how to breakdown work into manageable tasks were discussed by seven teams, and several strategies were identified. Some teams mirrored the work breakdown during the implementation phase to the system architecture designed for the system - *"The development process was split into sections to model our architecture. Modules allowed us to split our work load up evenly and allowed for minimal code/commit clashes as everyone was working on a different module"*. Other teams applied a divide and conquer approach - *"Most of the deliverables were shared between the group members, every member was given a specific task to do. Then we would complete our tasks individually and meet up at the library or other workspace to share the work and make fixes and help those that could not achieve their task since it was group work"*. In some cases, however, teams felt that this particular approach did not encourage idea sharing and collaboration.

Risk management. Only two teams discussed elements of risk management in their retrospectives. They reported on extensively researching potential risks at the beginning of the project, but then ignoring their analysis throughout the project - *"Extensive research into risk analysis was also done but we failed to follow through on some of the strategies laid out"*.

5.2.4. Environment

Attitude towards project. It was common for teams to begin with a highly positive mood with enthusiasm gradually decreasing throughout the duration of the project - *"We were all enthusiastic about our project at the beginning just like New Year's resolutions to get fit but as time passed our interests in the project development subsided, hence our attitude towards our work and each other wasn't always positive nevertheless our team always focused on finishing the work"*. This, coupled with the fact that students worked with colleagues they have not worked with before, caused significant stress. The reactions to stress varied. Some teams (5 teams) reported a disorganised end, and in extreme cases, complete abandonment of the project (2 teams). Other teams, however, remained focused on the end result and turned the challenge into an opportunity - *"This group was formed outside of our circle of friends, therefore it helped us to learn to work with new people, of different backgrounds and experiences, it pushed us out of our comfort zone but eventually, everyone coalesced to work even through disagreement at times"*.

Communication. All teams were required to use one communication platform for remote communication, with many teams opting for Slack. However, not all teams were comfortable with this, and 10% of the teams decided to use additional communication platforms (eg. Whatsapp). Instead of helping, this approach made remote communication more challenging - *"Other members would then be unclear on who was attending the meeting and when the meeting was taking place due to the organisation taking place on more than one communication platform"*. Teams reported various scenarios for communication breakdowns, including delays in replying to messages or forgetting to update the rest of the team of someone's actions. Remote communication was found to be difficult and face-to-face meetings were generally preferred. To manage poor attendance in meetings, students used communication platforms to update the absent team members on the discussions held during the meeting.

Coordination. Coordinating efforts were discussed in 11 retrospectives, the focus being on breakdowns scenarios. These ranged from time wasted due to poor task synchronisation and some tasks accidentally ending up being completed by two team members with inconsistencies resulting when individual contributions were merged into a single deliverable. In one case, two different systems were developed in parallel - *"We ended up developing two systems, one that was a mobile application and one that was a web application. When the systems had been finished it was made clear that they did not link well with each other"*.

Table 4
Top recurring codes reported in the retrospective documents and the map between the codes and the scenarios students rated at the beginning of the project.

Code	Theme	No. (%) of teams	Sc.
Delivered a different software system than initially planned	Product delivered	14 (40%)	
Lack of commitment from team members	Team members	12 (35%)	S7
Difficulty in managing other course deadlines at the same time	Time management	11 (32%)	
Version control management strategies	Development process	10 (30%)	
Work left to the last minute and rushed	Time management	10 (30%)	S6
Organising regular meetings	Meetings	9 (26%)	
Role allocation based on experience or preference	Role allocation	9 (26%)	
Team members leaving the team half way through the project	Team members	9 (26%)	S9
Elaborate and unrealistic initial plan	The plan	8 (23%)	
No clear leader/lack of leadership	Leadership	7 (20%)	S3
Strategies around documenting meetings	Meeting	6 (18%)	
Early realised that more expertise is needed to complete project	Team expertise	6 (18%)	
Disorganised/intense end of project	Communication	5 (15%)	
Same task done by two people	Coordination	5 (15%)	S5
Setting own milestones	Meeting deadlines	5 (15%)	
Tracking minutes of meetings	Meetings	5 (15%)	
Teams explore collective experience & expertise	Team expertise	5 (15%)	
Initial project plan ignored	The plan	4 (12%)	
Split team into sub-groups	Role allocation	4 (12%)	
Conflicting visions for the project	Shared understanding	3 (9%)	S8
I am the only one organising meetings	Meetings	3 (9%)	S1
Uneven contributions from team members	Individual contributions	3 (9%)	
Overestimating technical ability of team	Team expertise	3 (9%)	

Conflict resolution. While conflicts were not a common theme across the retrospectives, teams mostly discussed the strategies they used to overcome them. Such strategies included voting and debates - *"Seeing each other in person also meant that we could discuss issues that we encountered and talk it through and agree upon the best possible course of action"*.

Team spirit. The team spirit was reported by several teams as being the make-or-break element of the entire project. For some teams, a "friendly atmosphere accommodated with a balance of work ethic within the group" was the key to the success of the project overall. Team members learned from each other and supported each other - *"Since we weren't working by ourselves we could motivate each other to stay on topic and produce work at an efficient rate"*. A well working team, however, was often the result of a conscious effort to build one - *"To effectively build a functioning team, we learned the need to maintain discipline, set standards, appoint roles, encourage communication within the team, inspire motivation and encourage a sense of purpose"*. For other teams, however, the lack of synergy made the teamwork experience stressful and led to irregular meetings, work done at the last minute, and a general attempt to postpone or even ignore the work that was required to complete the project.

5.3. RQ3 - expectations meet reality

To answer RQ3, we analysed how the expectations students had at the beginning of the year match the realities they described in the retrospective accounts. For this, we identified the top 15% most recurring codes (Table 4) used in the thematic analysis of the retrospectives. These codes are summaries of the tokenised quotations depicted from the retrospective documents. While each quotation was associated with a code, a code was usually used to summarise quotations referring to the same issue. We then match the most recurring codes to the nine scenarios whose likelihood students rated at the beginning of the year.

Out of the nine scenarios students were presented with at the beginning of the year, only S7 ("We have not heard from a team member for a long time") was rated with an average higher than 3 (3.04) on a scale of 1 (very unlikely) to 5 (very likely). The scenario rated the least likely was S9 ("A member of the team decided we should use a specific method or language and then s/he stopped attending the team meetings"), with an average rating of 1.94. Based on the thematic analysis of the retrospective accounts, however, all scenarios but one were identified in the top 15% of the most recurring codes used. Lack of commitment from team members (S7) was reported by 35% of the teams, while a third of the teams reported leaving work to the last minute and rushing deliverables (S6). Nine of the teams mentioned team members leaving the team half way through the project (S9), and 20% of the teams discussed the lack of leadership in their team (S3). Lack of coordination which led to the same task being done by two team members (S5) was reported by 15% of the teams. Conflicting visions for the project (S8) and team members feeling burdened by the sole responsibility of the project or parts of the work (S1) were reported by 9% of the teams.

Additional breakdown scenarios teams reported included experiencing difficulties in managing other course deadlines scheduled at the same time, drafting elaborate and unrealistic initial plans which became useless or required significant overhaul as the project developed, facing difficulties in managing an intense or disorganised end of the project, and having to manage uneven contributions from team members with some team members contributing the bulk of the work. We note that while some of these scenarios seem to stem to the sequential software development put in place (eg. "Initial project plan ignored" or "Elaborate and unrealistic initial plan"), most of the scenarios have the potential to occur irrespective of the software development process put in place.

6. Discussion and practical implications

We investigate the expectations students have when embarking on an undergraduate software engineering team project and the realities they face. We discuss some of the breakdown scenarios students experience and the strategies they put in place to overcome them. We found a gap between the students' expectations and the reality they face with respect to teamwork, and propose four guidelines to narrow or bridge this gap.

Focus on the start of the project. The teams met the start of the project with great enthusiasm and with the expectation that none of the breakdown scenarios identified in past years are likely to happen to them. However, this expectation is often skewed, and the initial enthusiasm gradually fades when they face teamwork

breakdown challenges. While it might appear the change was due to students feeling overwhelmed by the scale of the project or due to team members leaving the team or showing a gradually declining lack of commitment, we believe the issues arose due to little or no focus on key aspects at the start of the project. The first deliverable students submitted was a project plan. In most cases, the initial project plan focused only on scheduling the tasks required to complete the project, without due consideration of aspects such as logistics, getting to know the collaborative expertise of the team, the preferences and experience of each individual team member with software development, or the overall team environment. This led to most plans being infeasible, or ignored throughout the project. An additional consequence of not focusing on creating and maintaining a team environment from the outset was the gradual erosion of the level of engagement amongst the students. The start of the project should focus more on building trust, a shared understanding within the team, and the implications of non-engagement - both in terms of software engineering motivations (Sharp et al., 2009) and the impact that non-engagement has on the learning of other team members.

Make informal meetings part of the course. While scheduling meetings was one of the challenges students associated with teamwork before the start of the project, regular face-to-face meetings proved to be a predictor of success. However, not all teams took advantage of this, and they either met irregularly or simply gave up meeting all together or held their meetings online. We argue that informal team meetings need to be brought to the heart of the project by emphasising their importance and incentivising students to organise and attend such meetings. The teams which enforced registers to track attendance in team meetings found them useful, but due to their optional character they were at times ignored. Turning such mechanisms into requirements for the project and incentivising teams to keep track of attendance and document their meetings via minutes can potentially encourage teams to meet regularly, and take a more strategic approach to monitoring progress and individual contributions.

Emphasise the importance of leadership. Teams initially paid little attention to the concept of Leadership; it did not arise as a theme when analysing the student's expectations with respect to teamwork and the prospect of a lack of leadership seemed unlikely before the project began. In most cases, external factors such as deadlines or critical decisions, led the teams to turn their attention to this concept. Nonetheless, few teams took a strategic approach to appointing a leader. In hindsight, however, most teams saw the merit of focusing on assigning leadership roles and responsibilities early and strategically. Due to the gap between students' expectations in terms of teamwork and the realities of such an exercise, this realisation was the result of trial-and-error and not a initial consideration. It is, therefore, necessary to support teams in exploring the concept of leadership early in the process via leadership training, mentoring, or specifically tailored leadership exercises.

Develop strategies for dealing with opportunistic behaviour. A recurring source of frustration among teams was opportunistic behaviour on the part of one or more of the team members. This included limited contributions at the last minute, sporadic attendance in group meetings, or submitting unnecessary or poor quality work. The level of frustration is also explained by the gap between the expectations students had at the start of the project and the realities they faced throughout the project. While at the start of the project the prospect of team members leaving work to the last minute or someone in the team misunderstanding their tasks were seen as an unlikely scenario, this proved to be one the most recurring events students experienced during the project. We argue that more assessed milestones need to be enforced throughout the year. These can be organised as informal as peer review assessment or, if the resources are available, as periodic monitoring.

7. Conclusions

The aim of this article is to provide a better understanding of the gap between the expectations students have from teamwork at the start of a software development group project and the realities they describe at the end of the project. For this, we present the results of two studies we ran as part of a 27 week software engineering group project course involving 35 teams. As part of the first study, we analyse the results of a questionnaire delivered individually to students at the start of the project, and aiming to assess their experience and expectations in terms of teamwork. As part of the second study, we analyse the retrospective accounts submitted by the 35 teams at the end of the projects aiming to better understand the recurring issues teams struggled with throughout the project. In addition to the limitations described in Sections 4.3.2 and 4.4.2, we also note that the data was collected from students by the first author who also delivered the unit. As indicated by Isomöttönen et al. (2019), this can be a cause of bias.

Our work addresses three research questions. Below, we summarise our answers to each of them.

RQ1: What are the expectations students have in terms of teamwork at the start of a software development group project?

Students approach group projects on a positive note and with enthusiasm, not considering any of the common teamwork breakdown scenarios likely to happen as part of their project. They recognise teamwork as being a significant part of their previous experiences; however, they do not necessarily relate teamwork to software development. At the start of the project, students associate the benefits and challenges of teamwork with generic statements, none specifically related to collaborative software development.

RQ2: How do undergraduate students perceive the reality of teamwork after completing a software development group project?

Teams report issues in four areas of teamwork, namely the team and its members, the skills required to complete the project, the process put in place, and the team environment overall. The key predictors for success identified in the retrospectives are regular meetings, good work ethic on the part of all the team members, and an overall friendly team environment. Some of the recurring issues reported in the retrospectives include lack of commitment of one or more members of the team, difficulty in managing other course deadlines at the same time, work left to the last minute, and investing time in coming up with an elaborate and unrealistic plan at the start of the project.

RQ3: How do the expectations match the reality students face when developing software as a team?

While none of the scenarios presented to students at the start of the project were perceived as particularly likely to occur to their team, all but one scenario were identified in the top 15% of the most recurring issues reported in the retrospective accounts at the end of the project. The significant gap between the initial expectations and the realities described at the end of the project explains some of the breakdown scenarios identified as answer to RQ2.

We include several guidelines we believe can narrow or bridge this gap in software engineering group projects. First, set realistic expectations at the start of the project, and focus more on initial team setup. Second, bring informal meetings to the core of the project. Third, emphasise the role leadership plays in group projects, and support teams in exploring this concept early on in the process. Fourth, put in place strategies for dealing with opportunistic behaviour.

References

Ashaikh, R.A., Wilson, S., Jones, S., 2016. A persuasive social actor for activity awareness in learning groups. In: Proceedings of the 30th International BCS Human

Computer Interaction Conference: Fusion, HCI '16, BCS Learning & Development Ltd., pp. 45:1–45:12. doi:10.14236/ewic/HCI2016.45.
Bastarrica, M.C., Perovich, D., Samary, M.M., 2017. What can students get from a software engineering capstone course? In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET '17. IEEE Press doi:10.1109/ICSE-SEET.2017.15.
Briggs, J., 1991. Group projects in software engineering at york. In: SIGCSE Bull, Vol. 23, pp. 48–50. doi:10.1145/122697.122705.
Bruegge, B., Krusche, S., Alperowitz, L., 2015. Software engineering project courses with industrial clients. ACM Trans. Comput. Educ. (TOCE) 15 (4). doi:10.1145/2732155.
Clear, T., 2003. Documentation and agile methods: striking a balance. SIGCSE Bull. 35 (2), 12–13. DOI:10.1.1.596.7811
Clear, T., Beecham, S., Barr, J., Daniels, M., McDermott, R., Oudshoorn, M., Savickaite, A., Noll, J., 2015. Challenges and recommendations for the design and conduct of global software engineering courses: a systematic review. In: Proceedings of the 2015 ITiCSE on Working Group Reports, ITICSE-WGR '15. ACM, pp. 1–39. doi:10.1145/2858796.2858797.
Daniels, A.M.R., Cajander, M., von Konsky, B., 2011. Assessing professional skills in engineering education. In: Hamer, J., de Raadt, M. (Eds.), Australasian Computing Education Conference (ACE 2011), Vol. 114 of CRPIT. ACS, pp. 145–154.
Daniels, M., 2011. The contribution of open ended group projects to international student collaborations. ACM Inroads 1 (3), 79–84. doi:10.1145/1835428.1835446.
Delgado, D., Velasco, A., Aponte, J., Marcus, A., 2017. Evolving a project-based software engineering course: a case study. In: 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T), pp. 77–86. doi:10.1109/CSEET.2017.22.
Dillenbourg, P., 1999. What do you mean by collaborative learning? In: Dillenbourg, P. (Ed.) Collaborative-Learning: Cognitive and Computational Approaches. Emerald Group Publishing, pp. 1–19.
Dugan Jr., F.R., 2011. A survey of computer science capstone course literature. Comput. Sci. Educ. 21 (3). doi:10.1080/08993408.2011.606118.
Holmes, R., Allen, M., Craig, M., 2018. Dimensions of experientialism for software engineering education. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET '18. ACM, pp. 31–39. doi:10.1145/3183377.3183380.
Isomöttönen, V., Daniels, M., Cajander, A., Pears, A., Mcdermott, R., 2019. Searching for global employability: can students capitalize on enabling learning environments? ACM Trans. Comput. Educ. 19 (2), 11:1–11:29. doi:10.1145/3277568.
Molléri, J.S., Gonzalez-Huerta, J., Henningsson, K., 2018. A legacy game for project management in software engineering courses. In: Proceedings of the 3rd European Conference of Software Engineering Education. ECSEE'18, pp. 72–76. doi:10.1145/3209087.3209094.
Oliveira, I., Tinoca, L., Pereira, A., 2011. Online group work patterns: how to promote a successful collaboration. Comput. Educ. 57 (1), 1348–1357. doi:10.1016/j.compedu.2011.01.017.
Peters, A., Hussain, W., Cajander, A., Clear, T., Daniels, M., 2015. Preparing the global software engineer. In: 2015 IEEE 10th International Conference on Global Software Engineering, pp. 61–70. doi:10.1109/ICGSE.2015.20.
Raibulet, C., Fontana, F.A., 2018. Collaborative and teamwork software development in an undergraduate software engineering course. J. Syst. Softw. 144, 409–422. doi:10.1016/j.jss.2018.07.010.
Richard, G.T., Kafai, Y.B., Adleberg, B., Telhan, O., 2015. StitchFest: diversifying a college Hackathon to Broaden participation and perceptions in computing. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15. ACM, pp. 114–119. doi:10.1145/2676723.2677310.
Schilling Jr., W.W., Sebern, M.J., 2013. Teaching software engineering: an active learning approach. Comput. Educ. J. 23 (1), 13–24.
Sedelmaier, Y., Landes, D., 2017. Practicing soft skills in software engineering: a project-based didactical approach. In: Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications, pp. 232–252. doi:10.4018/978-1-5225-3923-0.ch011.
Sharp, H., Baddoo, N., Beecham, S., Hall, T., Robinson, H., 2009. Models of motivation in software engineering. Inf. Softw. Technol. 51 (1), 219–233. doi:10.1016/j.infsof.2008.05.009.
Steeples, C., Unsworth, C., Bryson, M., Goodyear, P., Riding, P., Fowell, S., Levy, P., Duffy, C., 1996. Technological support for teaching and learning: computer-mediated communications in higher education (CMC in HE). Comput. Educ. 26 (1), 71–80. doi:10.1016/0360-1315(95)00082-8.
Stol, K.-J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, pp. 120–131. doi:10.1145/2884781.2884833.
Vanhanen, J., Lehtinen, T.O., Lassenius, C., 2018. Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project. J. Syst. Softw. 137, 50–66. doi:10.1016/j.jss.2017.11.021.
Wiggberg, M., 2010. Computer Science Project Courses: Contrasting Students' Experiences with Teachers' Expectations. Uppsala Universitet Ph.d. thesis.

Dr Claudia Iacob is a Senior Lecturer in Computer Science at University of Portsmouth, United Kingdom. She earned her PhD in Computer Science from University of Milan. Her research focused on user generated online content and its role in software development, as well as software engineering education and training.

Dr Shamal Faily is a Principal Lecturer in Systems Security Engineering at Bournemouth University. He earned his DPhil in Computer Science from the University of Oxford. His research explores how software can be designed to ensure it remains secure and usable when used in different contexts, as well as cybersecurity education and training.