A systematic literature review of capstone courses in software engineering

Tenhunen, Saara Maija

2023-07

Tenhunen , S M , Männistö , T , Luukkainen , M & Ihantola , P 2023 , ' A systematic literature review of capstone courses in software engineering ' , Information and Software Technology , vol. 159 , no. 2 . https://doi.org/10.1016/j.infsof.2023.107191

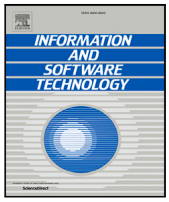
http://hdl.handle.net/10138/357159
https://doi.org/10.1016/j.infsof.2023.107191

Information and Software Technology 159 (2023) 107191

Contents lists available at ScienceDirect

Information and Software Technology

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



A systematic literature review of capstone courses in software engineering

Saara Tenhunen *, Tomi Männistö *, Matti Luukkainen, Petri Ihantola

The University of Helsinki, Finland

ARTICLE INFO

Keywords:
Capstone
Project course
Computer science education
Software engineering education

ABSTRACT

Context: Tertiary education institutions aim to prepare their computer science and software engineering students for working life. While much of the technical principles are covered in lower-level courses, team-based capstone courses are a common way to provide students with hands-on experience and teach soft skills.
Objective: This paper explores the characteristics of project-based software engineering capstone courses presented in the literature. The goal of this work is to understand the pros and cons of different approaches by synthesising the various aspects of software engineering capstone courses and related experiences.
Method: In a systematic literature review for 2007–2022, we identified 127 articles describing real-world capstone courses. These articles were analysed based on their presented course characteristics and the reported course outcomes.
Results: The characteristics were synthesised into a taxonomy consisting of duration, team sizes, client and project sources, project implementation, and student assessment. We found out that capstone courses generally last one semester and divide students into groups of 4–5 where they work on a project for a client. For a slight majority of courses, the clients are external to the course staff and students are often expected to produce a proof-of-concept level software product as the main end deliverable. The courses generally include various forms of student assessment both during and at the end of the course.
Conclusions: This paper provides researchers and educators with a classification of characteristics of software engineering capstone courses based on previous research. We also further synthesise insights on the reported course outcomes. Our review study aims to help educators to identify various ways of organising capstones and effectively plan and deliver their own capstone courses. The characterisation also helps researchers to conduct further studies on software engineering capstones.

1. Introduction

Universities and other tertiary education institutions should provide their students with sufficient skills and abilities before the students enter working life. In software engineering related programmes, this entails having an understanding of the common principles and theory in computer science [1,2] and technical competencies and knowledge demanded by the industry [3,4]. Any recent graduate should also be able to apply this technical knowledge in practice [1].

While much of the technical knowledge and theories are covered in lower-level courses, many institutions hold team-based capstone courses to ensure students are ready to apply the knowledge in a workplace environment. A "capstone course" usually means a course that finishes an academic degree [5]. The main goal of a project-based capstone course is to provide hands-on experience in applying the tools, techniques, principles and best practices that are taught more theoretically in previous courses [6–8]. Capstones are also regarded as crucial in teaching students the necessary soft skills such as teamwork [9,10], verbal and written communication [11], time management [12], problem solving [7] and project management [13]. In computer science (CS) and software engineering (SE) programmes, capstone courses generally last one or two semesters, and they include assigning students into teams and having them work on various kinds of software engineering projects [5,14,15]. In these projects, they are expected to experience stages of the software development life-cycle from requirements elicitation to software maintenance [9].

Given the general acceptance of capstones as a practical way of teaching industry-relevant skills, a high number of institutions include a capstone course in their curriculum. This has resulted in a great deal of research and experience reports done on capstones and their outcomes. In order to provide a coherent and compact view of software engineering capstones, this research synthesises the current body of knowledge on the topic in a systematic manner. We believe that such a review gives educators valuable knowledge when they plan and

* Corresponding authors.
E-mail addresses: saaraten@gmail.com (S. Tenhunen), tomi.mannisto@helsinki.fi (T. Männistö), matti.luukkainen@helsinki.fi (M. Luukkainen), petri.ihantola@helsinki.fi (P. Ihantola).

https://doi.org/10.1016/j.infsof.2023.107191
Received 18 October 2022; Received in revised form 26 February 2023; Accepted 2 March 2023
Available online 9 March 2023

Table 1
Searches for systematic reviews on software engineering capstones.

Database	Search term	Hits
Scopus	TITLE-ABS-KEY(software AND engineering AND capstone AND literature AND review)	10
Scopus	TITLE-ABS-KEY(software AND engineering AND capstone AND review)	61
Scopus	TITLE-ABS-KEY(education AND "software engineering" AND literature AND mapping)	40
Scopus	TITLE-ABS-KEY(project AND course AND software AND engineering AND systematic AND review)	20
Scopus	TITLE-ABS-KEY("computer science" AND capstone AND literature AND review)	8
Scopus	TITLE-ABS-KEY(software AND engineering AND project AND course AND systematic AND literature AND review)	17
Scopus	TITLE-ABS-KEY("software engineering" AND education AND literature AND review)	182
Google Scholar	Software engineering capstone systematic review	20 300
Google Scholar	Software engineering capstone characteristics	31 400
Google Scholar	Computer science capstone literature review	53 400
Google Scholar	Capstone literature review	94 800

Table 2
Systematic reviews of software engineering capstones.

Title	Year	Ref	Course characteristics examined in the survey
A survey of computer science capstone course literature	2011	Dugan Jr. [21]	Course-related: models, learning theories, goals, topics, student evaluation, evaluation. Project-related: software process models, phases, type, documentation, tools, groups, instructor administration.
Designing the IT capstone course	2019	Martin [22]	Course duration, learning of new skills, project identification and selection, teams sizes, team formation, followed methodologies, assessment of learning outcomes, team and project supervision[a]
A review of literature on assessment practices in capstone engineering design courses: Implications for formative assessment	2006	Trevisan et al. [23]	Connection to student achievement

[a]For the survey by Martin [22], these are characteristics, which would have been examined in the actual survey.

modify their own capstone courses. Researchers can also benefit from a systematic review of capstones to conduct further comparative studies on the impact of the varying course forms.

This study is organised as follows. The next section focuses on the previous literature reviews on SE capstones, as well as general characteristics of such courses. Section 3 describes the research questions and the related methods, including how the articles were selected. Section 4 presents the results of the literature review. The main findings and their validity are discussed in Section 5. Finally, Section 6 concludes the research and provides suggestions for future research.

2. Previous work

2.1. Systematic literature reviews of SE capstones

Many literature reviews have been written in the general area of software engineering education (SEE). Usually, they focus on specific sub-areas of SEE such as teaching methods in software engineering [16], practical approaches to SEE [17], trends in SEE [18,19] or teaching global software engineering [20].

As the focus of this research is especially on project-based capstone courses in software engineering, we carefully sought any earlier systematic reviews done on them. The search was conducted on May 17th, 2022, first in the citation database Scopus and secondly in Google Scholar. Table 1 lists the search terms used in both databases. All search results produced by Scopus were checked to see whether they include an SLR of SE capstones, whereas for Google Scholar, each search produced tens of thousands of hits, so we went through the first 20 pages of each search (200 hits). At this point, the results started to become highly irrelevant and often repetitive. Based on our search, we believe that the three review papers presented in Table 2 are the ones that have been published so far on this topic. Table 2 also presents the characteristics of capstone courses each of these reviews investigated. Next, we will briefly present these studies and discuss the necessity of this review.

Dugan Jr. [21] presents a survey done on the literature related to undergraduate computer science capstone courses. The survey is comprehensive, comprising of 200 papers on the subject and summarising them under two major themes: course issues and project issues (Table 2). Out of these, course issues include aspects related to the general course organisation, such as course models, learning theories present in the course and student evaluation. Project issues, on the other hand, categorise and describe the projects and how they are implemented. The category includes issues such as software process phases, project type and documentation of the projects.

Martin [22] has provided an abstract of a systematic literature review for designing an IT capstone course. The review plans to provide answers to several questions relating to capstone course design, such as the optimal team size for project teams, identifying and selecting suitable projects and determining the correct duration for the course (Table 2). We are not aware that the research proposed in the abstract would have been completed.

Trevisan et al. [23] have performed a systematic review on the assessment practices in capstone engineering design courses. They were especially interested in discovering the extent to which classroom assessment has received attention in the capstone literature. The paper included 32 journal articles and conference proceedings presenting varying assessment techniques and their use.

In addition to the presented three literature reviews, a study on the dimensions of SE and CS capstone courses has been conducted by Burge and Gannod [24]. Said dimensions are roughly divided into two groups: project dimensions such as customer identity and development dimensions such as project type and source code visibility. The purpose of their study is to provide a framework for analysing capstone courses, especially in terms of risk and realism. While their categorisation is versatile, their study does not, however, include a thorough systematic literature review. The categorisation presented is more of an experience-based proposal, and therefore the study is left out of Table 2.

To the best of our knowledge, there is no extensive, recent literature review done on software engineering capstone courses. A survey conducted by Dugan Jr. [21] is comprehensive but dated to 2011 and therefore does not cover the large number of primary studies published in the past decade. It also does not provide any statistics of the course characteristics, which would enable educators or researchers to assess how common some aspect in reality is. Trevisan et al. [23] provide a review on capstone literature, but it is limited to continuous assessment

Table 3
ACM/IEEE recommendations for SE capstones.

CR #	Recommendation
CR 1	The project should span a full academic year, giving students adequate time to reflect upon experiences and retry solutions as appropriate.
CR 2	Where possible, this should preferably be undertaken as a group project. If such factors as assessment make this difficult, it is essential that there should be a separate group project of substantial size.
CR 3	Where possible, a project should have a "customer" other than the supervisor so that the student gains fuller experience with product development life-cycle activities.
CR 4	A project should have some form of implementation as its end deliverable so that the students can experience a wide set of software development activities and adequately evaluate these experiences. Theory-based projects such as the development of formal specifications is therefore inappropriate for this role.
CR 5	Evaluation of project outcomes should go beyond concept implementation ("we built it, and it worked" [26]), using walkthroughs, interviews, or simple experiments to assess the effectiveness and limitations of the deliverables.
CR 6	Assessment of a capstone project should consider how effectively software engineering practices and processes have been employed, including the quality of student reflection on the experience, and not be based only on the delivery of a working system.

techniques and is dated to 2006. Martin [22] aims to provide a systematic literature review on IT capstone design and characteristics, but as of now, the paper has not proceeded beyond the original abstract. In light of this, current research does not provide an up-to-date view of how SE capstone courses generally are organised and with what kind of outcomes. Such a view on the software engineering capstones would not only provide educators with important information for planning their own capstone courses but also give researchers a basis for performing comparative studies on these courses.

2.2. Background: Capstone course characteristics

ACM/IEEE Curriculum Guidelines for Software Engineering (SE) Degree Programmes [2] view the capstone as an essential element of a SE degree programme and state that the main goal of a capstone course is to ensure that the curriculum has a significant real-world basis. According to ACM/IEEE [2], incorporating real-world elements into the curriculum is necessary to enable effective learning of software engineering skills and concepts. The ACM/IEEE Curriculum Guidelines for Computer Science (CS) degree programmes [1] align with these views and state that all graduates of CS programmes should have been involved in at least one substantial project. Such projects should challenge students by being integrative, requiring evaluation of potential solutions and working on a larger scale than typical course projects. For students, a project-based capstone course typically represents a culmination of their studies and is one of the last milestones before graduation [1,25]. Indeed, since the 1970s, hundreds of primary studies have been written on this large, final-year project course [21].

The ACM/IEEE [2] also lists a set of key recommendations that a capstone course should follow. The recommendations are listed word by word in Table 3. We decided to use these recommendations as the basis for formulating our research questions. They give a general outline of capstone courses and therefore provide a valid starting point for the categorisation done in this research.

Thus according to these guidelines, there are some basic characteristics that capstone courses have. They can be characterised as long and substantial projects (CR1, CR2) that should preferably be completed in a team (CR2). Projects should have customers (CR3) for whom the students are expected to deliver some form of real implementation at the end of the course (CR4). Students should therefore engage in real software development activities and not just complete simple, theory-based assignments provided by the teacher (CR4). Evaluation of the project outcomes should focus not only on the fact that the project "works", but also assess the deliverables on how well they have been completed (CR5). Finally, the focus of the course and its assessment should be on software engineering practices and processes and students should give adequate opportunities to reflect on the experience (CR6). The next section describes in more detail the process of how we derived the research questions based on these basic characteristics.

3. Research questions and method

3.1. Research questions

Characteristics of capstone courses (described in Section 2.2) can be achieved in many ways. The main goal of this research was to understand these differences in how capstone courses are implemented in universities and other tertiary education institutions, and thus provide a holistic view over the various capstone course implementations.

We decided to use the ACM/IEEE Curriculum Guidelines for Undergraduate SE Degree Programmes [2] as the basis for starting to explore these characteristics. The recommendations are listed in Table 3. We are not aware of any study covering all the aspects mentioned in the ACM/IEEE recommendations.

Related to CR1, we were interested in the duration of the courses and what rationale articles provide for choosing a specific course duration, if any:

RQ1 What is the duration of SE capstone courses, and what advantages or disadvantages are related to a certain duration?

Related to CR2, we wanted to find out if these projects are conducted in teams and what is the rationale behind choosing a certain team size. Team formation strategies were left out of scope as there are pre-existing literature reviews done on that matter alone [27]. The research question was therefore formed as:

RQ2 What team sizes do SE capstone courses have, and how are team sizes justified?

Based on the ACM/IEEE recommendations (i.e., CR3), a project should have a customer other than the teacher of the course. An alternative approach to bringing an outside view to a project is to outsource project topics. Thus, our third research question, *how are the project and client sourcing handled in SE capstone courses (RQ3)*, was divided into two sub-questions:

RQ3.1 Who acts as the client for capstone projects?

RQ3.2 How are the ideas for projects sourced?

Related to CR4, we wanted to find out: *How are the projects in capstone courses implemented (RQ4)*. We wanted to uncover what students do in these courses and therefore, we looked into the actual project implementation. As 'project implementation' can mean a multitude of things, we decided to divide this research question into smaller, more concrete sub-questions:

RQ4.1 What artefacts are students expected to produce on capstone courses?

RQ4.2 What is the software life-cycle adopted on these courses?

RQ4.3 How are the implementation technologies chosen for capstone projects?

With RQ4.1 we aimed to find out what students actually produce in these courses and whether any software is being developed. RQ4.2 helped us to find out if the capstone project is as integrative experience on software engineering practices as curriculum guidelines [2,25] suggest. Finally, finding out how educators make the choices for implementation technologies and what implications these choices have, gave some insight into project implementation.

Since CR5 and CR6 consider assessment, our last research question was formulated as: *How is the student assessment conducted on SE capstone courses (RQ5)*. As assessment can be divided into continuous feedback and final grading, RQ5 was also split into two:

RQ5.1 How are the students assessed at the end of SE capstone courses?

RQ5.2 How are the students guided through continuous assessment, if at all, during SE capstone courses?

The rationalisation here was that we wanted to uncover whether the evaluation is based on a multitude of factors like [2] suggests and whether students are given adequate possibilities to reflect on their experiences [28].

In order to get a comprehensive representation of how project-based capstone courses are generally organised, relevant articles were searched. One could argue that the characteristics and any organisational details of these courses could be derived from the web pages of universities and other tertiary institutions. However, we wanted not only to produce a list of characteristics such as the duration and workload of the courses but also to reveal more about the contents of these courses. An important part of the research was also to provide educators with insights related to the various characteristics. Without any evaluation or assessment of the chosen structure and characteristics, this would have been impossible to achieve.

3.2. Search strategy

The method used in this study follows the SLR method by Kitchenham and Charters [29]. The initial data collection was done by finding relevant sources from scientific databases: Scopus, ACM Digital Library, IEEE Xplore and ScienceDirect. Some preliminary searches were conducted on these databases to find out to which extent sources use the word "capstone" and its synonyms when describing large, degree-culminating project courses in software engineering-related programmes. It turned out that the term "capstone" is well-known and widely used in relevant literature. It was also used by Dugan Jr. [21] in their earlier work. Therefore, the first search string was simply constructed as:

```
software AND capstone
```

In order to have a complete picture of the project course landscape in software engineering, a second search was performed using the second search string:

```
software AND 'project course'
```

This was deemed necessary as not all sources had the word "capstone" present in the metadata even though they clearly were describing courses relevant to this research. Searches with the two search strings were conducted sequentially in each database.

Dugan Jr. [21] used "software engineering course" as another search term, but we did not want to limit ourselves to the SE discipline, as relevant software-related courses might be presented, for instance, in computer science. Using only the words "software" and "course" on the other hand, provided too many irrelevant hits. Scopus alone produced nearly 30 000 hits of which only a small fraction would have been relevant to our study.

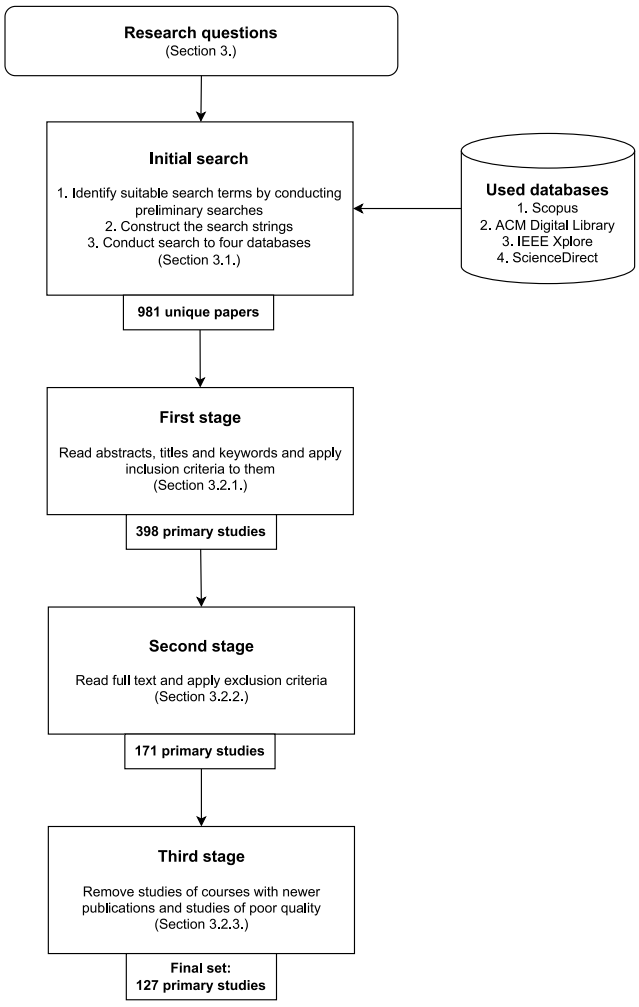


Fig. 1. Search strategy.

A total of 981 unique papers were found after combining the papers found from all four databases using the search strings and removing duplicates. The databases were searched on June 11th and June 12th 2022, one after the other, starting with Scopus, moving on to ACM Digital Library, followed by ScienceDirect and finishing with IEEE Xplore. As the search fields and filters are slightly different in each of these databases, the search strings were adjusted to match each specific set-up. They were, however, kept semantically the same across the searches. Exact search strings and initial search results are listed in Table 4. As we wanted to identify current ways of organising capstone courses, the searches in all four databases were limited to the years 2007 to the search day in June 2022. This time period was regarded as sufficiently long to provide a holistic view of the current capstone courses. It overlaps with Dugan Jr. [21] by a few years but also uncovers 11 years of research and reporting done on the area that has not been systematically reviewed since. Three stages of selection were applied to this initial set, after which 127 articles remained. Fig. 1 summarises the search and selection process, and the following subsections will describe it in greater detail.

3.3. Paper selection

The paper selection was conducted from the initial set of 981 sources by the first author (Fig. 1). The details of inclusion and exclusion are explained next.

Table 4
Initial search results.

Database	Search strings	Hits
Scopus	TITLE-ABS-KEY (software AND capstone)	762
Scopus	TITLE-ABS-KEY (software AND "project course")	262
ACM Digital Library	[Title: software] AND [Title: capstone]	24
ACM Digital Library	[Keywords: software] AND [Keywords: capstone]	32
ACM Digital Library	[Abstract: software] AND [Abstract: capstone]	130
ACM Digital Library	[Title: software] AND [Title: "project course"]	6
ACM Digital Library	[Keywords: software] AND [Keywords: "project course"]	7
ACM Digital Library	[Abstract: software] AND [Abstract: "project course"]	44
ScienceDirect	(TITLE ABS KEY: SOFTWARE CAPSTONE)	22
ScienceDirect	(TITLE ABS KEY: SOFTWARE PROJECT COURSE)	12
IEEE Xplore	("All Metadata":Software) AND ("All Metadata":capstone)	223
IEEE Xplore	("All Metadata":software) AND ("All Metadata":"project course")	86

3.3.1. The first stage—Inclusion criteria

The titles, abstracts and keywords of the initial papers were read and evaluated against the inclusion criteria presented below (IC1–IC3). After the first stage, 398 papers remained.

IC1 The title or abstract strongly hints that the article presents frameworks or case studies of software engineering capstones or other large, project-based courses in software engineering.

IC2 Based on the title or abstract, the article describes real experiences of implementing a software engineering capstone course.

IC3 The title or abstract indicates that the article assesses the outcomes of the course or its characteristics.

The first inclusion criterion was developed to set the focus on software engineering courses in particular. A large number of the articles in the initial set were ruled out due to the first criterion (IC1). The papers were found to research, for instance, mechanical engineering courses, which were out of the scope of this research. We also wanted to rule out any purely hypothetical papers, where the researchers show no course that follows the frameworks or structures presented. The second inclusion criterion (IC2) aimed to ensure that all included papers would present a real-world course. The final inclusion criterion (IC3) was generated so that all papers would also evaluate the outcomes of the various course implementations.

3.3.2. The second stage—Exclusion criteria

The second stage was performed on the 398 papers remaining from the first stage. Any article that, based on reading the *full paper*, met at least one of the presented exclusion criteria was excluded at this stage. After this selection, 171 articles remained for the final evaluation. The used exclusion criteria were:

EC1 The length of the article is less than four pages.

EC2 The article is not published in conference proceedings or as a journal article.

EC3 The article does not have full text available in English.

EC4 The article turned out not to describe a software engineering capstone course in a tertiary institution.

EC5 The article is not able to provide answers to most of the research questions.

Exclusion criteria from EC1 through EC3 aimed to ensure that the article was of sufficient quality. According to Kitchenham and Charters [29] workshop proceedings often do not provide sufficient input for the purposes of an SLR. Additionally, quite many of the papers that were first published as short workshop proceedings or abstracts were also found to have a conference proceeding or a journal article published later on. EC3 relates to the language skills of the authors as well as the status of English as the primary language in software engineering-related research. These exclusion criteria led to some papers being rejected before reading their entire content.

For exclusion criteria EC4–EC5, the content of the article was examined more carefully and, in most cases, read in its entirety to make a justified decision. Exclusion criteria EC4 and EC5 relate to our research goal. For instance, many articles were found to describe courses in computer engineering or mini-projects conducted prior to SE capstones which meant that they were out of the scope of this research and excluded based on EC4. Most of the papers left out during this stage met EC4. As for EC5, some articles were, for example, found to describe a whole curriculum with capstone courses playing only a minor part in the research, and they could therefore not provide answers to our research questions. A number of articles also evaluated a tool, method or framework relevant to the software engineering industry, not the capstone course itself. In many of these articles, the capstone course presented the researchers merely a convenient way of gaining study participants, which is why they did not fit this research. This led them to be excluded due to EC5.

3.3.3. The third stage—Removal of duplicates and articles of poor quality

The third stage was included mainly to rule out any duplicate data and articles of poor quality from our research. After the third stage, the final set of 127 papers remained.

Duplicate data—All 171 articles remaining after the second stage describe real-life software engineering capstones. As educators often like to modify their courses over time to find the best ways of teaching, articles here too reflect on the changes done to the courses. Some authors also have written multiple articles based on the same capstone course. In such cases, the most recent article was chosen. Similarly, if the authors describe several instances of the course in one paper, the principal characteristics of the most recent course instance were chosen for the data extraction. Choosing the latest instance of each course stems from the goal of this research to synthesise the current state of knowledge on capstone implementations. In addition, the decision of whether two descriptions of the same course are different enough for them to be included as their own capstone courses would have been too ambiguous and open for interpretation. Kitchenham and Charters [29] also state that it is important not to include multiple publications of the same data, as it would seriously bias any results. Due to this procedure, 42 articles were removed from the final set.

Quality assessment—In addition to inclusion/exclusion criteria, Kitchenham and Charters [29] state that it is critical to perform a quality assessment on the primary studies. We also conducted such an assessment and used it to ensure that our final data set is of sufficient quality. At this stage, two articles were filtered out. Any tables or graphs presented from here on do not include these two excluded articles, and therefore represent the final set of 127 articles.

Table 5 lists the set of questions used by Dybå and Dingsøyr [30], Ali et al. [31] and Mahdavi-Hezavehi et al. [32] which we also used to determine the quality of the articles. Originally the questions were

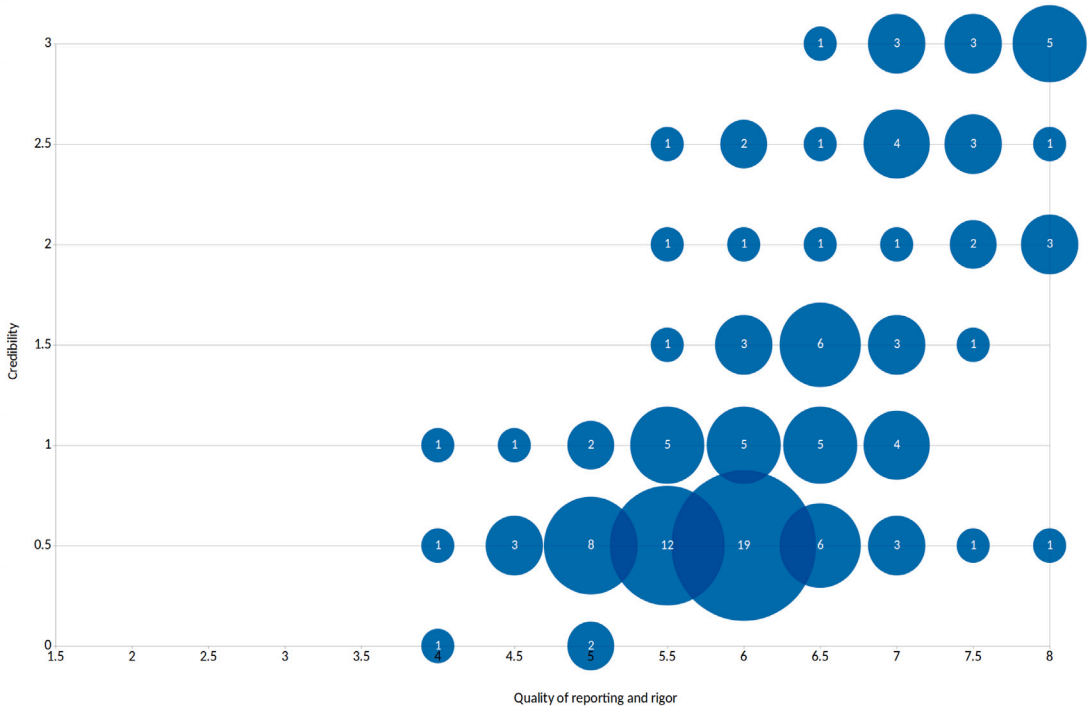


Fig. 2. Quality scores of the final set of articles.

Table 5
Questions for quality assessment.

No.	Question
Q1	Is there a rationale for why the study was undertaken?
Q2	Is there an adequate description of the context (e.g. industry, laboratory setting, products used, etc.) in which the research was carried out?
Q3	Is there a justification and description for the research design?
Q4	Has the researcher explained how the study sample (participants or cases) was identified and selected, and what was the justification for such selection?
Q5	Is it clear how the data was collected (e.g. through interviews, forms, observation, tools, etc.)?
Q6	Does the study provide a description and justification of the data analysis approaches?
Q7	Has 'sufficient' data been presented to support the findings?
Q8	Is there a clear statement of the findings?
Q9	Did the researcher critically examine their own role, potential bias and influence during the formulation of research questions, sample recruitment, data collection, and analysis and selection of data for presentation?
Q10	Do the authors discuss the credibility of their findings?
Q11	Are limitations of the study discussed explicitly?

supposed to be graded on a dichotomous ("Yes" = 1 or "No" = 0) scale [30], but we decided to use a three-point scale of "Yes" (=1), "To some extent" (=0.5) and "No" (=0). This three-point scale has also been adopted by Ali et al. [31] and Mahdavi-Hezavehi et al. [32] and allowed us better to assess the articles where authors only provided some answers to the question. The two articles filtered out had a quality score of less than 4.

In our assessment, we decided to group the first eight questions to represent the quality of reporting and rigour of the articles and the final three questions to represent the credibility of evidence, similarly to Ali et al. [31]. The grouped scores are presented in Fig. 2 and individual scores for each article can be found at https://github.com/article-additions. Regarding the quality of reporting, the selected articles performed fairly well. It was mostly clear how the data had been collected, and the relevance of the study was explicitly discussed. However, the aspect most articles were lacking was providing justifications either for the sample selection or research designs. Regarding the credibility of evidence, the articles performed fairly poorly. Interestingly, many of the otherwise well-established articles did not include a section for explicitly discussing the limitations of the study or the author's role in data and sample selection. This is indicated in the low averages of the credibility category.

3.3.4. Overview of the final papers

The three stages taken resulted in 127 articles, published between 2007 and 2022 (June). Research activity in this area has been fairly steady over the years, as depicted in Fig. 3. It is worth noting, that we conducted the searches in June 2022, thus covering the year 2022 only partially. Also, as explained in Section 3.3.3, 42 earlier articles, which otherwise would have been valid for this research, were excluded from the final set as there was a newer article of the same course available. This procedure skews the publication year distribution towards the end of the scale. The figure also shows the distribution by article type. In total, 73% of articles were published in conference proceedings and 27% were journal articles.

All the articles included for further analysis are listed in Table 6, and referenced later in this section with their publication ID in the table (i.e., S1–S127.)

3.4. Data extraction and synthesis

After applying the article selection process, the properties presented in Table 7 were extracted from the remaining 127 articles to a common datasheet. Table 7 defines how each extracted field relates to the research questions of this study.

Table 6
Included sources for data extraction.

ID	Author(s)	Year	Title	Source title
S1	Marzolo, P., Guazzaloca, M., Ciancarini, P.	2021	"Extreme Development" as a Means for Learning Agile	International Conference on Frontiers in Software Engineering
S2	Tan, J., Jones, M.	2008	A case study of classroom experience with client-based team projects	Journal of Computing Sciences in Colleges
S3	Wong, W., Pepe, J., Stahl, J., Englander, I.	2013	A collaborative capstone to develop a mobile hospital clinic application through a student team competition	Information Systems Education Journal
S4	Tappert, C. C., Stix, A.	2011	A decade review of a masters-level real-world-projects capstone course	Info. Systems Educators Conf., ISECON 2011
S5	Gotel, O., Kulkarni, V., Say, M., Scharff, C., Sunetnanta, T.	2009	A global and competition-Based model for fostering technical and soft skills in software engineering education	22nd Conference on Software Engineering Education and Training, CSEE&T 2009
S6	Scott, A., Kreahling, W., Holliday, M., Barlowe, S.	2017	A holistic capstone experience: Beyond technical ability	18th Annual Conference on Information Technology Education
S7	Koolmanojwong, S., Boehm, B.	2013	A look at software engineering risks in a team project course	26th International Conference on Software Engineering Education and Training, CSEE&T 2013
S8abcd	Braught, G., et al.	2018	A multi-institutional perspective on H/FOSS projects in the computing curriculum	ACM Transactions on Computing Education
S9	Mertz, J., Quesenberry, J.	2019	A scalable model of community-based experiential learning through courses and international projects	2018 World Engineering Education Forum - Global Engineering Deans Council, WEEF-GEDC 2018
S10	Bloomfield, A., Sherriff, M., Williams, K.	2014	A Service Learning Practicum capstone	45th ACM technical symposium on Computer science education
S11	Brazier, P., Garcia, A., Vaca, A.	2007	A software engineering senior design project inherited from a partially implemented software engineering class project	37th Annual Frontiers in Education Conference - Global Engineering
S12	Morales-Trujillo, M.E., Galster, M., Gilson, F., Mathews, M.	2021	A Three-Year Study on Peer Evaluation in a Software Engineering Project Course	IEEE Transactions on Education
S13	Liang, Z., Chapa-Martell, M.A.	2019	A Top-Down Approach to Teaching Web Development in the Cloud	IEEE International Conference on Teaching, Assessment, and Learning for Engineering, TALE 2018
S14	Murphy, C., Sheth, S., Morton, S.	2017	A Two-Course Sequence of Real Projects for Real Customers	Conference on Integrating Technology into Computer Science Education, ITiCSE 2017
S15	Rusu, A., Rusu, A., Docimo, R., Santiago, C., Paglione, M.	2009	Academia-academia-industry collaborations on software engineering projects using local-remote teams	40th ACM Technical Symposium on Computer Science Education, SIGCSE'09
S16	Stettina, C.J., Zhao, Z., Back, T., Katzy, B.	2013	Academic education of software engineering practices: towards planning and improving capstone courses based upon intensive coaching and team routines	26th International Conference on Software Engineering Education and Training, CSEE&T 2013
S17	Venson, E., Figueiredo, R., Silva, W., Ribeiro, L.C.M.	2016	Academy-industry collaboration and the effects of the involvement of undergraduate students in real world activities	IEEE Frontiers in Education Conference, FIE 2016
S18	Eloe, N., Hoot, C.	2020	Accommodating Shortened Term Lengths in a Capstone Course using Minimally Viable Prototypes	IEEE Frontiers in Education Conference, FIE 2020
S19	Schneider, J.-G., Eklund, P.W., Lee, K., Chen, F., Cain, A., Abdelrazek, M.	2020	Adopting industry agile practices in large-scale capstone education	42nd International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2020
S20	Ye, H.	2009	An academia-industry collaborative teaching and learning model for software engineering education	21st International Conference on Software Engineering and Knowledge Engineering, SEKE 2009
S21	Demuth, B., Kandler, M.	2017	An Approach for Project Task Approximation in a Large-Scale Software Project Course	30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017
S22	Ellis, H.J.C.	2007	An assessment of a self-directed learning approach in a graduate web application design and development course	IEEE Transactions on Education
S23	Anslow, C., Maurer, F.	2015	An experience report at teaching a group based agile software development project course	46th ACM Technical Symposium on Computer Science Education
S24	Bareiss, R., Katz, E.	2011	An exploration of knowledge and skills transfer from a formal software engineering curriculum to a capstone practicum project	24th IEEE-CS Conference on Software Engineering Education and Training, CSEE&T 2011
S25	Stephenson, B., James, M., Brooke, N., Aycock, J.	2016	An Industrial Partnership Game Development Capstone Course	17th Annual Conference on Information Technology Education

(*continued on next page*)

Table 6 (*continued*).

S26	Bell, J.T., Prabhu, A.	2015	An innovative approach to Software Engineering term projects, coordinating student efforts between multiple teams over multiple semesters	IEEE Frontiers in Education Conference, FIE 2014
S27	Vasilevskaya, M., Broman, D., Sandahl, K.	2015	Assessing large-project courses: Model, activities, and lessons learned	ACM Transactions on Computing Education, TOCE
S28	von Konsky, B.R., Ivins, J.	2008	Assessing the capability and maturity of capstone software engineering projects	Tenth conference on Australasian computing education - Volume 78
S29	Fontao, A., Gadelha, B., Junior, A.C.	2019	Balancing Theory and Practice in Software Engineering Education - A PBL, toolset based approach	IEEE Frontiers in Education Conference, FIE 2019
S30	Harding, T.	2007	Benefits and struggles of using large team projects in capstone courses	ASEE Annual Conference and Exposition
S31	Engelsma, J. R.	2014	Best practices for industry-sponsored CS capstone courses	Journal of Computing Sciences in Colleges
S32	Matthies, C., Teusner, R., Hesse, G.	2019	Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts	IEEE Frontiers in Education Conference, FIE 2018
S33	Ziv, H., Patil, S.	2010	Capstone project: From software engineering to "Informatics"	23rd IEEE Conference on Software Engineering Education and Training, CSEE&T 2010
S34	Anderson, Ruth E.; Borriello, Gaetano; Martin, Hélène; Black, Leonard	2009	Capstone projects as community connectors	Journal of Computing Sciences in Colleges
S35	Paasivaara, M., Vanhanen, J., Lassenius, C.	2019	Collaborating with industrial customers in a capstone project course: The customers' perspective	IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019
S36	Adams, R., Kleiner, C.	2016	Collaboration support in an international computer science capstone course	International Conference on Social Computing and Social Media
S37	Watkins, K.Z., Barnes, T.	2010	Competitive and agile software engineering education	IEEE SoutheastCon, SoutheastCon 2010
S38	Gustavsson, H., Brohede, M.	2019	Continuous assessment in software engineering project course using publicly available data from GitHub	15th International Symposium on Open Collaboration, OpenSym 2019
S39	Hadfield, Steven M.; Jensen, Nathan A.	2007	Crafting a software engineering capstone project course	Journal of Computing Sciences in Colleges
S40	Rong, G., Shao, D.	2012	Delivering software process-specific project courses in tertiary education environment: Challenges and solution	25th IEEE Conference on Software Engineering Education and Training, CSEE&T 2012
S41	Nguyen, D.M., Truong, T.V., Le, N.B.	2013	Deployment of capstone projects in software engineering education at Duy Tan university as part of a university-wide project-based learning effort	Learning and Teaching in Computing and Engineering, LaTiCE 2013
S42	Lago, P., Schalken, J., Vliet, H.V.	2009	Designing a multi-disciplinary software engineering project	22nd IEEE Conference on Software Engineering Education and Training, CSEE&T 2009
S43	Angelov, S., de Beer, P.	2017	Designing and applying an approach to software architecting in agile projects in education	Journal of Systems and Software
S44	Anderson, R.E., Kolko, B.	2011	Designing technology for resource-constrained environments: A multidisciplinary capstone sequence	Frontiers in Education, FIE 2012
S45	Leilde, V., Ribaud, V.	2017	Does Process Assessment Drive Process Learning? the Case of a Bachelor Capstone Project	30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017
S46	Brown, Q., Lee, F., Alejandre, S.	2009	Emphasizing soft skills and team development in an educational digital game design course	4th International Conference on the Foundations of Digital Games, FDG 2009
S47	Takala, T. M., Malmi, L., Pugliese, R., Takala, T.	2016	Empowering students to create better virtual reality applications: A longitudinal study of a VR capstone course	Informatics in Education
S48	Marques, M., Ochoa, S.F., Bastarrica, M.C., Gutierrez, F.J.	2018	Enhancing the Student Learning Experience in Software Engineering Project Courses	IEEE Transactions on Education
S49	De Souza, R.T., Zorzo, S.D., Da Silva, D.A.	2015	Evaluating capstone project through flexible and collaborative use of Scrum framework	Frontiers in Education Conference, FIE 2015
S50	Vu, J.H., Frojd, N., Shenkel-Therolf, C., Janzen, D.S.	2009	Evaluating test-driven development in an industry-sponsored capstone project	6th International Conference on Information Technology: New Generations, ITNG 2009
S51	Laplante, P.A., Defranco, J.F., Guimaraes, E.	2019	Evolution of a graduate software engineering capstone course - A course review	International Journal of Engineering Education
S52	Lederman, Timoth C.	2010	Evolution of capstone-courses in software engineering a finishing school	Journal of Computing Sciences in Colleges
S53	Delgado, D., Velasco, A., Aponte, J., Marcus, A.	2017	Evolving a Project-Based Software Engineering Course: A Case Study	30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017

(*continued on next page*)

Table 6 (*continued*).

S54	Widyani, Yani	2013	Experience in Software Development Project Course	Procedia Technology
S55	Ras, Eric and Carbon, Ralf and Decker, Bj̀orn and Rech, J́org	2007	Experience Management Wikis for Reflective Practice in Software Capstone Projects	IEEE Transactions on Education
S56	Schorr, R.	2020	Experience Report on Key Success Factors for Promoting Students' Engagement in Software Development Group Projects	4th IEEE World Conference on Engineering Education, EDUNINE 2020
S57	Longstreet, C. Shaun; Cooper, Kendra	2013	Experience report: A sustainable serious educational game capstone project	CGAMES'2013 USA
S58	Dupuis, R., Champagne, R., April, A., Séguin, N.	2010	Experiments with Adding to the Experience that Can be Acquired from Software Courses	7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010
S59	Burge, J.	2007	Exploiting Multiplicity to Teach Reliability and Maintainability in a Capstone Project	20th IEEE Conference on Software Engineering Education and Training, CSEE&T 2007
S60	Marshall, L., Pieterse, V., Thompson, L., Venter, D.M.	2016	Exploration of Participation in Student Software Engineering Teams	ACM Transactions on Computing Education, TOCE
S61	Ganci, A., Ramnath, R., Ribeiro, B., Stone, R.B.	2011	Exploring collaboration between computer science engineers and visual communication designers in educational settings	13th International Conference on Engineering and Product Design Education, E&PDE 2011
S62	Burden, H., Steghöfer, J.-P., Hagvall Svensson, O.	2019	Facilitating entrepreneurial experiences through a software engineering project course	41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019
S63	Basholli, A., Baxhaku, F., Dranidis, D., Hatziapostolou, T.	2013	Fair assessment in software engineering capstone projects	6th Balkan Conference in Informatics
S64	Magana, A. J., Seah, Y. Y., Thomas, P.	2018	Fostering cooperative learning with Scrum in a semi-capstone systems analysis and design course	Journal of Information Systems Education
S65	Sievi-Korte, O., Systä, K., Hjelsvold, R.	2015	Global vs. local – Experiences from a distributed software project course using agile methodologies	Frontiers in Education, FIE 2015
S66	Hebig, R., Ho-Quang, T., Jolak, R., Schröder, J., Linero, H., Ågren, M., Maro, S.H.	2020	How do students experience and judge software comprehension techniques?	28th International Conference on Program Comprehension
S67	Verdicchio, Michael	2021	Hurricanes and pandemics: an experience report on adapting software engineering courses to ensure continuity of instruction	Journal of Computing Sciences in Colleges
S68	Włodarski, R., Poniszewska-Marańda, A., Falleri, J.-R.	2022	Impact of software development processes on the outcomes of student computing projects: A tale of two universities	Information and Software Technology
S69	Izu, Cruz	2018	Improving Outcomes for a Masters Capstone IT Project	IEEE International Conference on Teaching, Assessment, and Learning for Engineering, TALE 2018
S70	Flowers, J.G.	2008	Improving the Capstone project experience: a case study in software engineering	46th Annual Southeast Regional Conference on XX
S71	Gannod, Gerald C.; Bachman, Kristen M.; Troy, Douglas A.; Brockman, Steve D.	2010	Increasing alumni engagement through the capstone experience	Frontiers in Education, FIE 2010
S72	Zilora, S.J.	2015	Industry-emulated projects in the classroom	16th Annual ACM Conference on Information Technology Education, SIGITE 2015
S73	Spichkova, M.	2019	Industry-oriented project-based learning of software engineering	24th International Conference on Engineering of Complex Computer Systems, ICECCS 2019
S74	Carvalho, J.A., Sousa, R.D., Sá, J.O.	2010	Information systems development course: Integrating business, IT and IS competencies	2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments
S75	Palacin-Silva, M.V., Seffah, A., Porras, J.	2018	Infusing sustainability into software engineering education: Lessons learned from capstone projects	Journal of Cleaner Production
S76	Kumar, S., Wallace, C.	2015	Instruction in software project communication through guided inquiry and reflection	Frontiers in Education, FIE 2015
S77	Zeid, A.	2012	Integrating international students' contests with computer science capstone: Lessons learned and best practices	Frontiers in Education, FIE 2012
S78	Lundqvist, K., Ahmed, A., Fridman, D., Bernard, J.-G.	2019	Interdisciplinary Agile Teaching	Frontiers in Education, FIE 2019
S79	Santoso, H.B., Lawanto, O., Purwandari, B., Isal, R.Y.K., Fitriansyah, R.	2018	Investigating Students' Metacognitive Skills while Working on Information Systems Development Projects	7th World Engineering Education Forum, WEEF 2017

(*continued on next page*)

Table 6 (*continued*).

S80	Christensen, E.L., Paasivaara, M.	2022	Learning Soft Skills through Distributed Software Development	International Conference on Software and System Processes and Internation Conference on Global Software Engineering
S81	Rout, Terence P.; Seagrott, John	2007	Maintaining High Process Capability in a Student Project Course	20th Conference on Software Engineering Education & Training, CSEE&T 2007
S82	Rodriguez, G., Soria, A., Campo, M.	2016	Measuring the Impact of Agile Coaching on Students' Performance	IEEE Transactions on Education
S83	Linhoff, J., Settle, A.	2009	Motivating and evaluating game development capstone projects	4th International Conference on Foundations of Digital Games
S84	Haddad, H.M.	2013	One-semester CS capstone: A 40-60 teaching approach	10th International Conference on Information Technology: New Generations, ITNG 2013
S85	Fan, Xiaocong	2018	Orchestrating Agile Sprint Reviews in Undergraduate Capstone Projects	Frontiers in Education, FIE 2018
S86	Fagerholm, F., Vihavainen, A.	2013	Peer assessment in experiential learning: Assessing tacit and explicit skills in agile software engineering capstone projects	Frontiers in Education, FIE 2013
S87	Vasankari, T., Majanoja, A.-M.	2019	Practical Software Engineering Capstone Course – Framework for Large, Open-Ended Projects to Graduate Student Teams	Internation Conference on Computer Supported Education
S88	Karunasekera, S., Bedse, K.	2007	Preparing software engineering graduates for an industry career	20th Conference on Software Engineering Education & Training, CSEE&T 2007
S89	Weerawarana, S.M., Perera, A.S., Nanayakkara, V.	2012	Promoting creativity, innovation and engineering excellence: A case study from Sri Lanka	IEEE International Conference on Teaching, Assessment, and Learning for Engineering, TALE 2012
S90	Fornaro, R.J., Heil, M.R., Tharp, A.L.	2007	Reflections on 10 years of sponsored senior design projects: Students win-clients win!	Journal of Systems and Software
S91	Roach, S.	2011	Retrospectives in a software engineering project course: Getting students to get the most from a project experience	24th IEEE-CS Conference on Software Engineering Education and Training, CSEE&T 2011
S92	Mäkiaho, P., Poranen, T.	2018	Risks management in software development capstone projects	19th International Conference on Computer Systems and Technologies
S93(a,b)	MacKellar, B. K., Sabin, M., Tucker, A.	2013	Scaling a framework for client-driven open source software projects: A report from three schools	Journal of Computing Sciences in Colleges
S94	Yuen, T.T.	2015	Scrumming with educators: Cross-departmental collaboration for a summer software engineering capstone	International Conference on Learning and Teaching in Computing and Engineering, LaTiCE 2015
S95	Isom̀ott́onen, V., Daniels, M., Cajander, Å., Pears, A., McDermott, R.	2019	Searching for global employability: Can students capitalize on enabling learning environments?	ACM Transactions on Computing Education
S96	Maxim, B.	2008	Serious games as software engineering capstone projects	ASEE Annual Conference and Exposition
S97	Krogstie, B.R., Divitini, M.	2009	Shared timeline and individual experience: Supporting retrospective reflection in student software engineering teams	22nd Conference on Software Engineering Education and Training, CSEE&T 2009
S98	Johns-Boast, L., Flint, S.	2013	Simulating industry: An innovative software engineering capstone design course	Frontiers in Education, FIE 2013
S99	Boti, E., Damasiotis, V., Fitsilis, P.	2021	Skills Development Through Agile Capstone Projects	International Conference on Frontiers in Software Engineering
S100	Paiva, S.C., Carvalho, D.B.F.	2018	Software creation workshop: A capstone course for business-oriented software engineering teaching	XXXII Brazilian Symposium on Software Engineering
S101	Saeedi, K., Visvizi, A.	2021	Software development methodologies, HEIs, and the digital economy	Education Sciences
S102	Smith, T., Cooper, K.M.L., Longstreet, C.S.	2011	Software engineering senior design course: Experiences with agile game development in a capstone project	International Conference on Software Engineering
S103	Jaccheri, L., Sindre, G.	2007	Software engineering students meet interdisciplinary project work and art	11th International Conference on Information Visualisation, IV 2007
S104	Krusche, S., Dzvonyar, D., Xu, H., Bruegge, B.	2018	Software Theater—Teaching Demo-Oriented Prototyping	ACM Transactions on Computing Education, TOCE
S105	Budd, A.J., Ellis, H.J.C.	2008	Spanning the gap between software engineering instructor and student	Frontiers in Education, FIE 2008
S106	Decker, A., Egert, C.A., Phelps, A.	2016	Splat! er, shmup? A postmortem on a capstone production experience	Frontiers in Education, FIE 2008
S107	Kerbs, R.	2007	Student teamwork: A capstone course in game programming	Frontiers in Education, FIE 2007

(*continued on next page*)

Table 6 (*continued*).

S108	Tadros, Ibrahem; Hammami, Samir; Al-Zoubi, Khaled	2008	Systems Development Projects	3rd International Conference on Information and Communication Technologies: From Theory to Applications
S109	Jarzabek, S.	2013	Teaching advanced software design in team-based project course	26th IEEE International Conference on Software Engineering Education and Training, CSEE&T 2013
S110	Lu, Baochuan; DeClue, Tim	2011	Teaching agile methodology in a software engineering capstone course	Journal of Computing Sciences in Colleges
S111	Cagiltay, N.E.	2007	Teaching software engineering by means of computer-game development: Challenges and opportunities	British Journal of Educational Technology
S112	Tafliovich, A., Caswell, T., Estrada, F.	2019	Teaching software engineering with free open source software development: An experience report	Annual Hawaii International Conference on System Sciences
S113	Paasivaara, M., Lassenius, C., Damian, D., Raty, P., Schroter, A.	2013	Teaching students global software engineering skills using distributed Scrum	35th International Conference on Software Engineering, ICSE 2013
S114	Khmelevsky, Y.	2016	Ten years of capstone projects at Okanagan College: A retrospective analysis	21st Western Canadian Conference on Computing Education
S115	Mahnič, V.	2015	The capstone course as a means for teaching agile software development through project-based learning	World Transactions on Engineering and Technology Education
S116	Broman, D., Sandahl, K., Baker, M.A.	2012	The company approach to software engineering project courses	IEEE Transactions on Education
S117	Khakurel, J., Porras, J.	2020	The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students	32nd IEEE Conference on Software Engineering Education and Training, CSEE&T 2020
S118	Iacob, C., Faily, S.	2020	The impact of undergraduate mentorship on student satisfaction and engagement, teamwork performance, and team dysfunction in a software engineering group project	51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020
S119	Hoar, R.	2014	The real world web: How institutional IT affects the delivery of a capstone web development course	19th Western Canadian Conference on Computing Education, WCCCE 2014
S120	Yue, K. B., Damania, Z., Nilekani, R., Abeysekera, K.	2011	The use of free and open source software in real-world capstone projects	Journal of Computing Sciences in Colleges
S121	Isomʾottʿonen, V., Kʾarkkʿainen, T.	2008	The value of a real customer in a capstone project	21st Conference on Software Engineering Education and Training, CSEE&T 2008
S122	Mohan, S., Chenoweth, S., Bohner, S.	2012	Towards a better capstone experience	43rd ACM Technical Symposium on Computer Science Education, SIGCSE'12
S123	Rico, D.F., Sayani, H.H.	2009	Use of agile methods in software engineering education	Agile Conference, AGILE 2009
S124	Tribelhorn, B., Nuxoll, A.M.	2021	Using Agile and Active Learning in Software Development Curriculum	ASEE Virtual Annual Conference and Exposition
S125	McDonald, J., Wolfe, R.	2008	Using computer graphics to foster interdisciplinary collaboration in capstone courses	Journal of Computing Sciences in Colleges
S126	Ju, A., Hemani, A., Dimitriadis, Y., Fox, A.	2020	What agile processes should we use in software engineering course projects?	51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020
S127	Bastarrica, M.C., Perovich, D., Samary, M.M.	2017	What can students get from a software engineering capstone course?	39th IEEE/ACM International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017

Table 7
Data extraction form.

Identifier	Field	RQ
F1	Title	Metadata
F2	Author(s)	Metadata
F3	Year	Metadata
F4	Publication venue	Metadata
F5	Duration of the course	RQ1
F6	Course workload	RQ1
F7	Team sizes	RQ2
F8	Clients	RQ3.1
F9	Project sources	RQ3.2
F10	Artefacts produced	RQ4.1
F11	Project phases	RQ4.2
F12	Technologies	RQ4.3
F13	Student assessment	RQ5
F14	Outcomes of the course	RQ1–RQ5
F15	Quality score	QA

Values F1–F4 were extracted for basic documentation purposes. Items F5–F14 concern the course and its organisation presented in the article. Two of the reviewed articles present multiple separate capstone courses from different institutions. For these two articles (S8 and S93 in Table 6), the items F5–F14 were extracted for each course they presented. For F5–F14, we were not only interested in quantifying these characteristics into statistics but also in providing implications of different course design choices. Therefore, if the article stated, for example, that they had a two-semester capstone course because it provided students adequate time to learn, we recorded both of these information pieces: the quantifiable duration and any such insight relating to the characteristic. This enabled us to analyse and discuss the course characteristics better in Section 4. A data-driven thematic analysis was applied to synthesise the qualitative data extracted as part of F5–F14 [33].

F5 and F6 were considered essential in assessing the general workload and duration of the course from the student's perspective. Few sources have given the duration of their course (F5) as months or

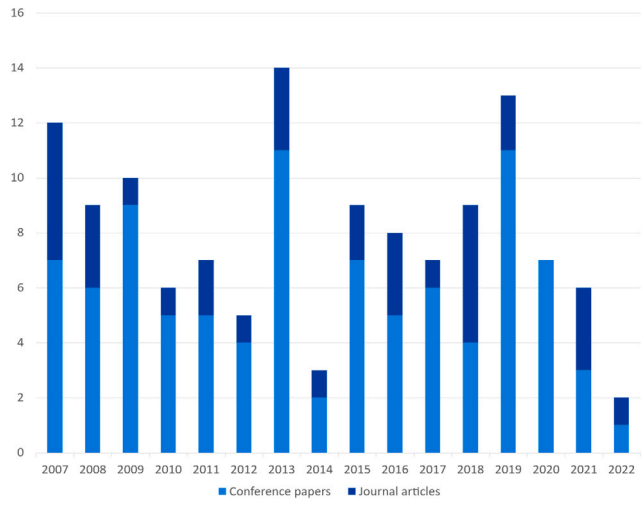


Fig. 3. Timeline and types of reviewed articles.

weeks. These were rounded to the nearest amount of semesters. Courses lasting less than 4 months were categorised as "less than one semester", 4 to 6 months as "one semester" and anything more than 6 but less than 10 months as "two semesters".

Team sizes (F7) included the number of students per team. The courses were also examined on whether the projects in the course were done for a client (F8). The client could be external to the course staff, the role of a client could be played by the course staff, and some projects did not have clients at all. Some sources present a mix of these categories, in which case, the source was labelled by the client category, which we thought was the most prevalent in the course.

We were also interested in how the project topics were generated (F9). Three main sources for projects were identified during the data extraction: course staff, external clients and the students themselves. The projects were also found to vary regarding whether the students were working on the same project idea or whether each team had their own initial problem.

We extracted all the artefacts students were expected to produce during the course (F10). This included the deliverables used for grading the course and the artefacts produced for project management reasons, as in most cases, it was hard to distinguish between them. Evidence of project phases (F11) was extracted to determine which software life-cycle activities were adopted on these courses. F12 describes the technologies used in the course. We found that most reviewed articles do not explicitly specify all the technologies used for the projects in their courses. Moreover, these technologies could potentially include any software technologies available. We, therefore, categorised these into two categories based on whether the main technology selections are made team-wise or all use a common technology stack.

We extracted information on how the students' learning process was assessed and improved throughout the course and how the students' progress and achievement were assessed at the end of the course (F13). The key outcomes in the article (F14) were also extracted to assess the advantages and disadvantages of the presented capstone characteristics. F15 was extracted as presented in Section 3.3.3 for quality assessment and article filtering.

4. Results and analysis

This section presents the quantitative analysis and qualitative outcomes of the capstone characteristics from the articles. The characterisation enables us to answer our research questions and ultimately helps educators when they are planning their capstone courses.

4.1. Duration (RQ1)

Regarding course characteristics, we first looked into the reported duration of these courses (F5). A clear majority of institutions conducted capstone courses that lasted one semester (Table 8). Interestingly, this is in conflict with the ACM/IEEE [2] recommendations for undergraduate capstone courses, which propose having capstones lasting the entire academic year. One reason given for a one-semester capstone was that not all curricula can afford a full-year implementation [S117]. Also, capstones were regarded as very labour-intensive for the teaching staff with many teams to manage and evaluate throughout the course [S39], [S112]. In some cases, students might have had full- or part-time work, which made it harder to arrange longer courses [S49], [S58], [S73]. Students also perceived two-semester courses as laborous [S73], [S109] and some even the one-semester ones [S23], [S41]. In order to provide an intensive and realistic experience, many of these courses took up at least half a work-week [S18], [S28], [S41], [S69], [S73], [S109], [S127]. This might have made other courses taken simultaneously suffer [S41], limiting the possibilities for an intensive, year-long capstone.

However, some educators who had experiences with both shorter and longer duration had shifted to longer duration since they felt it was impossible to reach the wanted skill coverage and depth in just a few months [S6], [S33]. One of the reviewed articles, which was authored by a student, strongly recommends that their course be lengthened into one academic year from one semester [S70]. One of the reviewed articles states that the change to a longer course version received overwhelmingly positive feedback from all the participating parties [S33]. Students were able to gain more hands-on experience in project management and in applying new and familiar tools. Industrial clients received more ambitious and polished products, and the course staff felt that the course learning objectives were finally truly met.

4.2. Team sizes (RQ2)

To find out how many students there were in project groups, we extracted the reported team sizes (F7) in Fig. 4. If a reviewed article refers to the course having teams of 4–5 students, this is thus reflected in both columns 4 and 5. Looking at Fig. 4, it is evident that capstone courses were almost always conducted as group projects. Only three institutions in our research allowed their capstone or senior project courses to be completed as single-student endeavours [S11], [S89], [S111].

Team sizes varied a great deal, ranging from 1 to 35. In very small groups, e.g., 2–3 students, the teams were not perceived to generate the dynamics and issues that are common in collaborative software development [S36], [S53], [S56], [S58]. One article stated that such a small team size does not present enough of a challenge [S36], and another that smaller groups are unable to complete substantial projects in a typical one-semester course [S53]. Having very small teams was also regarded as unmaintainable in large programmes with hundreds of students due to the extra organisational overhead each team causes [S19]. In the other extreme, larger groups with team sizes ranging from 7 up to 35 students were often found to be facing other kinds of problems, such as the inability to meet all together and other management and coordination issues [S30], [S36], [S39], [S53], [S78]. The "free-rider" problem was also perceived to be more common in larger teams, where it is possible for a few students to take a bigger responsibility in ensuring the overall success and the small contribution of others to go unnoticed [S9], [S56], [S58], [S69], [S87], [S121]. In larger teams, ensuring fair grading and an equal balance of work and responsibilities also required more attention from the course staff [S87].

One outlier course regarding team sizes is presented in [S106]. The course had 15 students working on the same game project in one team. The idea was to simulate what large-scale game development in

Table 8
Duration of capstone courses.

Category	Number of courses	Percentage	Article identifiers
Less than one semester	10	8%	S16, S18, S40, S46, S55, S61, S78, S94, S99, S113
One semester	87	66%	S1, S2, S3, S4, S5, S8b, S8d, S9, S11, S13, S15, S20, S21, S22, S23, S24, S25, S26, S27, S30, S31, S32, S34, S35, S36, S37, S38, S42, S43, S44, S45, S47, S48, S51, S53, S54, S56, S57, S58, S60, S62, S63, S64, S65, S66, S67, S68, S69, S70, S73, S74, S75, S76, S77, S79, S80, S81, S82, S83, S84, S86, S89, S90, S92, S93a, S93b, S95, S97, S100, S102, S103, S104, S105, S106, S107, S112, S115, S116, S117, S119, S120, S121, S123, S124, S125, S126, S127
Two semesters	32	24%	S6, S7, S8a, S8c, S10, S12, S14, S17, S19, S28, S33, S39, S41, S50, S52, S59, S71, S72, S85, S87, S88, S91, S96, S98, S101, S108, S109, S110, S111, S114, S118, S122
More than two semesters	1	1%	S49
Not specified	1	1%	S29

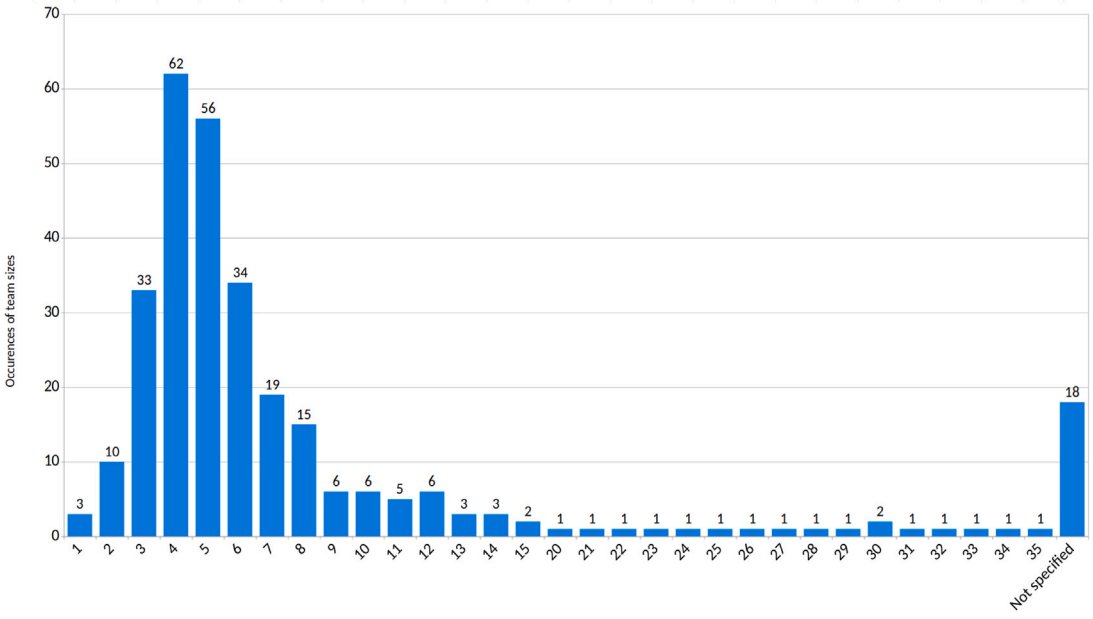


Fig. 4. Team sizes in capstone courses.

a diverse team feels like. The authors share that their approach was not entirely successful. In the aftermath of the course, it came up that some students wanted explicit direction while others felt that they wanted more autonomy and control. According to the authors, for the latter group of students, it was clear that they were uncomfortable following the leadership of the vision team and would have preferred to work on a project of their own design.

Most educators do seem to opt for the middle ground regarding team sizes and have 4 to 5 people working in a single group (Fig. 4). This size was perceived as the sweet spot, cancelling out the negatives of the two extremes [S36], [S52], [S53], [S56], [S58]. Students have also reported being satisfied with such a team size [S56]. Additional measures for combating non-productive group behaviour, such as free riding, have also been proposed. For example, peer reviews were successfully used to mitigate the risk of any opportunistic behaviour [S72], [S98]. Some periodic monitoring should also be done by the course staff to ensure working team dynamics [S69]. Both of these will be discussed further in Section 4.5.

4.3. Clients and project ideas (RQ3)

4.3.1. Clients (RQ3.1)

We also looked at who was in the role of a client for these projects (F8) and how the project ideas were sourced (F9). Almost half of the reviewed articles (42%) conducted their capstone courses without clients that are external to the course (Table 9). In these courses, the course staff acted as clients or Product Owners for the projects or alternatively, the student teams worked on their own and only reported progress regularly to the course staff. In some cases instructors played clients due to the difficulty of finding suitable clients [S6]. The institution's wish to own the intellectual property rights for the developed products was also found to put off potential clients [S6]. The time and effort it takes to handle multiple external clients and to ensure that their goals are aligned with the learning goals of the course might be unsustainable for some institutions [S56]. Having a complicated, cross-disciplinary course setup with technical and non-technical students also made some educators shy away from involving external clients in the course [S42]. Some reviewed articles also explain how the course outcomes were less predictable with multiple external clients. One article reports experiencing several cases when the project sponsors did not show up for the bi-weekly meetings with the students [S114]. Such client behaviour caused very low motivation in the student teams, and some capstone projects failed due to client unavailability. Other authors too have made similar observations and stress the importance of finding committed clients to ensure a good experience for the students [S3], [S23], [S78], [S122].

Table 9
Clients of capstone courses.

Category	Number of courses	Percentage	Article identifiers
Clients external to the course staff	76	58%	S2, S3, S4, S5, S7, S8a, S8d, S9, S10, S11, S14, S15, S17, S18, S19, S23, S24, S27, S28, S29, S30, S31, S33, S34, S35, S37, S38, S39, S41, S46, S48, S49, S50, S52, S55, S58, S59, S60, S61, S62, S63, S66, S67, S69, S71, S73, S78, S80, S81, S84, S85, S86, S87, S88, S90, S91, S92, S93a, S93b, S94, S96, S97, S98, S104, S108, S110, S112, S113, S114, S117, S120, S121, S122, S124, S126, S127
Course staff acts as clients	16	12%	S1, S6, S12, S21, S40, S43, S53, S54, S57, S68, S76, S82, S99, S115, S116, S123
No clients	38	29%	S8b, S8c, S13, S20, S22, S25, S26, S32, S36, S42, S44, S45, S47, S51, S56, S64, S65, S70, S72, S74, S75, S77, S79, S83, S89, S95, S100, S101, S102, S103, S105, S106, S107, S109, S111, S118, S119, S125
Not specified	1	1%	S16

Despite these risks, having real, external clients other than the course staff is recommended for both undergraduate and graduate capstones [2,25]. These clients can be from other units within the university [S7], [S9], [S14], [S52], [S58], [S86], [S87], [S94], local businesses [S7], [S9], [S86], [S87], [S98], [S110], [S122] or various non-profit organisations [S7], [S9], [S11], [S16], [S58], [S110]. Contacting graduates, who already work in the industry, was also a convenient way for finding external clients [S35]. Working closely with real-world clients often received highly positive feedback from students [S14], [S15], [S52], [S73], [S98], [S117] and organising staff alike [S14], [S35], [S84], [S98], [S117]. The motivation and commitment of students were found to increase when there was a real need behind the project [S9], [S14], [S15], [S35], [S66], [S73], [S84], [S121]. Having industry clients improved the students' technical and nontechnical skills and was perceived to prepare them well for the challenges they will face in their future work life [S14].

The collaboration was reported to have benefits for the clients too. Such benefits are specifically listed in a study by [S35], where the authors aimed to find out the reasons why clients participate in capstone courses. The reasons included getting a tailored software product, researching new technologies and, as a clear number one, recruitment. Recruiting could happen directly from the team or more indirectly by raising awareness of the organisation as a potential employer. Other articles report this phenomenon too; it was not uncommon for students to get hired by the industry partner who sponsored their capstone project [S15], [S28], [S35], [S73], [S84], [S114], [S121]. One of the articles reports that at least 60 out of a few hundred students secured full- or part-time job offers based on the project outcomes [S73]. Keeping the experience positive also for the clients made them come back with new project ideas and thus helped with client acquisition in the following years [S35]. Some institutions even managed to attract more external clients than there were student teams, which enabled them to collect a small fee from the participating clients [S35], [S121].

4.3.2. Project sources (RQ3.2)

We also looked into how the projects for these courses were sourced (F9). Three main ways for project sourcing were identified (Table 10). As most courses had multiple external clients, the project ideas in these courses were mainly derived from the customers' needs. In these cases, the organising staff often performed some pre-screening and scoping in collaboration with the clients to ensure that the expectations for the projects were realistic and that the project scopes suited the intended learning outcomes [S9], [S24], [S35], [S52], [S61], [S87], [S90], [S94], [S98], [S121]. Some of the reviewed articles recommend that capstone projects should generally not be on the critical path of any external organisation, as the course is intended to remain a safe learning place for the students [S35], [S52], [S87], [S90], [S121]. Some articles also emphasise that students are not supposed to be working for these clients but in collaboration with them [S9], [S52], [S87], [S121]. Thus, the students should have a say in how the software development will be done in the project.

For 17% of the courses, the students were the main source of project ideas. Students appeared to be more motivated if they got to choose the project idea themselves [S36], [S53]. At the end of the semester, the students had a strong sense of ownership towards the project, rather than a feeling that they had just completed one additional assignment [S36], [S53]. However, there are some potential pitfalls with this approach that educators should be aware of. One article states that students should not be allowed to bring project ideas from the companies they work at as it causes a conflict of interest for the student with the proposal and creates an unfair situation for the rest of the team [S114]. In some cases students were allowed to form their own teams and generate their own project ideas, even though this was not perceived to accurately reflect the situation in the students' future professional lives [S36]. One article states that if the project idea comes from the team all complexities associated with requirements elicitation and analysis are eliminated, making the experience less realistic [S73].

For 21% of the courses, the course staff provided the project specifications. Some educators assigned the same project idea to all the student teams [S40], [S45], [S72], [S82]. Moreover, some courses had only one project that the entire class worked on as one team [S106], [S112]. Having common project idea had the benefit of giving the course staff a consistent basis for teaching, grading [S40], [S59], [S72] and for providing technical assistance to the students [S40], [S53]. In such cases, all teams needed to deal with the same complexities, project management issues and technology demands, which made the experience more predictable [S72]. Having one project idea also opened up the possibility for competition amongst the teams, e.g., which team would create the best design and implementation [S5], [S30], [S37], [S42], [S54], [S72], [S106]. It also possibly allowed the course to focus more on the quality of the developed software [S5]. One article reports experiences with both approaches, having multiple project ideas and having only one project [S5]. In the end, the authors found it more rewarding to focus on doing one project really well rather than juggling multiple projects and obtaining several incomplete systems.

4.4. Project implementation (RQ4)

We also aimed to uncover information on how capstone projects were implemented and what students were expected to do in these courses. For this, we extracted various information on project implementation (F10–F12).

4.4.1. Produced artefacts (RQ4.1)

We were interested in finding out what kind of artefacts students were expected to produce on the course (F10). It was often difficult to determine which artefacts were used for the final student assessment (i.e. grading) and which were only produced to manage the project in

Table 10
Sources for projects in capstone courses.

Category	Number of courses	Percentage	Article identifiers
External stakeholders propose project ideas	81	62%	S2, S3, S4, S5, S7, S8a, S8b, S8c, S8d, S9, S10, S11, S14, S15, S17, S18, S19, S23, S24, S27, S28, S29, S30, S31, S33, S34, S35, S37, S38, S39, S41, S46, S48, S49, S50, S52, S55, S58, S59, S60, S61, S62, S63, S66, S67, S69, S71, S72, S73, S77, S78, S80, S81, S84, S85, S86, S87, S88, S90, S91, S92, S93a, S93b, S94, S96, S97, S98, S104, S108, S110, S112, S113, S114, S117, S120, S121, S122, S124, S125, S126, S127
Course staff provides project ideas	27	21%	S1, S6, S12, S13, S16, S20, S21, S25, S40, S42, S43, S45, S54, S57, S65, S68, S74, S75, S79, S82, S99, S105, S106, S109, S115, S116, S123
Students generate their own project ideas	22	17%	S22, S26, S36, S44, S47, S51, S53, S56, S64, S70, S76, S83, S89, S95, S100, S101, S102, S103, S107, S111, S118, S119
Not specified	1	1%	S32

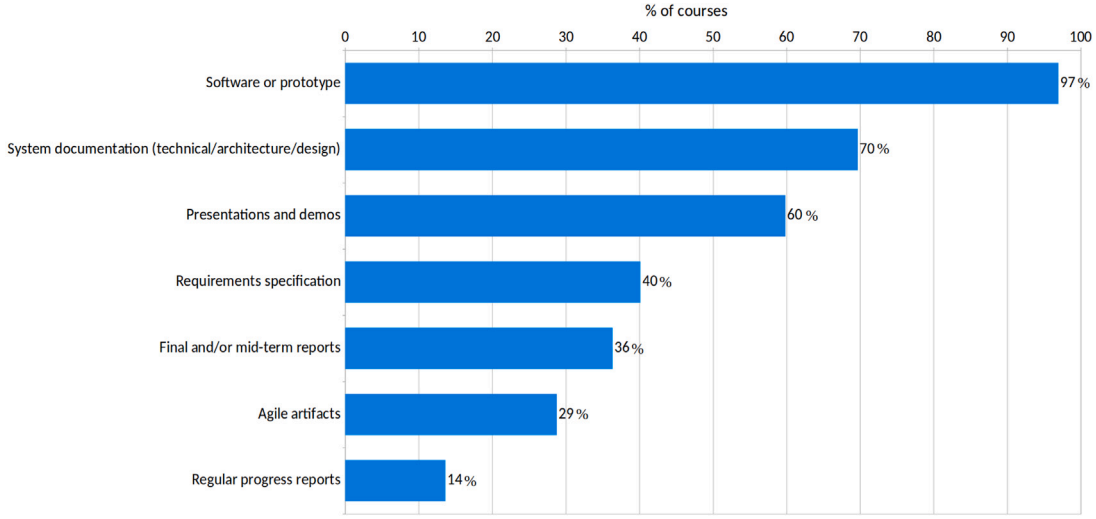


Fig. 5. Most commonly mentioned artefacts produced by students in capstone courses.

some way. Therefore we could not produce a list of graded artefacts. Fig. 5 provides a rough view of the explicitly mentioned artefacts that students were expected to produce in general.

We were, however, able to determine that all but three courses [S16], [S17], [S103] expected some form of a software prototype, a software product or source code as the only acceptable end deliverable. Most reviewed articles (70%) also mention requiring some documentation for the software project (Fig. 5). The actual number of courses that required documentation might be larger than reported here, as we only counted the times the study explicitly mentions that project documentation was done on the course. Reviewed articles at the beginning of our time range are often following more "plan-first" software development approaches, such as the waterfall model, and as a result, the quantity and detail of non-software artefacts were substantial [S2], [S11], [S22], [S70]. The documentation usually started heavily upfront with students producing project plans [S2], [S68], [S70], detailed designs [S2], [S22], [S52], [S68], architecture plans [S22], [S68] and test plans [S2], [S70]. In later, more agile courses, there was less evidence of extensive documentation and planning. With agile projects, the system documentation was largely developed as the project evolved [S14], [S65], [S94]. Students in these courses also often produced agile artefacts, such as product backlogs, sprint backlogs and burndown charts for project management and planning purposes [S6], [S65], [S94], [S124].

A large share of the reviewed articles (60%) explicitly mentions that students needed to present or demonstrate their projects to wider audiences than just the immediate project team. This taught students how to present and explain their work to a non-technical audience [S6]. The most common time for presentations was at the end of the semester, and typically these presentations were given at fairs or class sessions to which all stakeholders of the course were invited [S2], [S4], [S6], [S33], [S95]. The format of final presentations varied from poster sessions [S6], [S28], [S34], [S44] to live software demonstration sessions [S89], [S95] to different kinds of demo videos [S1], [S73], [S118]. Students were sometimes also expected to draw up a project proposal or a pitch and then present it to the teachers or the class before starting to work on the implementation [S87]. In these cases, the purpose was to offer the students a chance to practice their pitching skills [S100].

Students were often expected to detail software requirements during the course. These could be written down as Software Requirements Specifications before the implementation phase began [S30]. For agile projects, software requirements were often documented in an initial backlog with user stories, and the backlog was then updated as the project continued [S124]. Students were also quite often expected to write some form of a report at the end of the experience, either individually or as a group. The reports generally involved students reflecting on the learning done and development processes employed throughout the course [S6], [S19], [S21], [S24], [S27], [S41], [S63], [S65].

The balance between too little and too many artefacts was considered a delicate one. Having more documentation and deliverables

Table 11
Development technologies in capstone courses.

Category	Number of courses	Percentage	Article identifiers
Projects use common technologies	33	25%	S1, S5, S8a, S8d, S11, S13, S22, S32, S37, S38, S42, S43, S45, S46, S48, S53, S54, S57, S59, S72, S76, S79, S82, S83, S93b, S99, S105, S106, S107, S109, S112, S113, S124
Choices done primarily team-wise	65	50%	S2, S3, S4, S8b, S8c, S9, S10, S12, S14, S19, S24, S26, S29, S31, S33, S35, S36, S39, S44, S47, S50, S51, S52, S56, S58, S61, S62, S65, S66, S68, S69, S71, S73, S74, S75, S77, S80, S84, S85, S86, S87, S88, S89, S90, S92, S95, S96, S98, S100, S101, S102, S103, S104, S110, S111, S114, S116, S117, S118, S119, S120, S121, S122, S125, S127
Not specified	33	25%	S6, S7, S15, S16, S17, S18, S20, S21, S23, S25, S27, S28, S30, S34, S40, S41, S49, S55, S60, S63, S64, S67, S70, S78, S81, S91, S93a, S94, S97, S108, S115, S123, S126

presented teachers with more opportunities to grade and assess students' understanding of software development processes [S110]. On the other hand, more documentation meant less product which might not be in the interests of project clients [S49]. Indeed, some educators mandated only basic time tracking and reflective reporting from the students and left the majority of deliverables for the external client and team to decide [S24].

4.4.2. Project phases (RQ4.2)

We sought evidence of the project phases and software life-cycle adopted on these courses (F11). Software life-cycle models include, with varying frequency and order, phases such as requirements gathering or elicitation, planning and designing, developing, testing and maintaining the product [34]. Several of the reviewed articles report that for capstone courses the projects proceeded from ideas to robust proofs-of-concept or products with few core requirements implemented [S1], [S6], [S7], [S11], [S12] [S16], [S18], [S20], [S21], [S22], [S26], [S40], [S42], [S45], [S52], [S54], [S55], [S62], [S64], [S65], [S68], [S79], [S70], [S72], [S75], [S77], [S78], [S87], [S90], [S94], [S95], [S99], [S100], [S106], [S107], [S109], [S115], [S118], [S122], [S123], [S124], [S125]. Students thus got to experience the phases of planning, designing, developing and testing the products in these projects. In courses with clients, either external or internal, the students usually had to elicit the requirements from the clients (Table 10). However, sometimes the teachers provided students with ready-made feature or requirements lists [S12], [S21], [S45], [S79], [S82] and in some courses, students generated their own project proposals (Section 4.3.2). In these cases, students did not get to experience requirements gathering. Additionally, the projects generally did not include maintaining existing software in actual production use during the course. In some courses, some of the projects were production-ready at the end of the course, but these too were then handed over to the customer [S5], [S9], [S117]. This practice leaves students without the experience of working with existing products in the maintenance phase of the software life-cycle. Assigning students to contribute to Free and Open Source Software (FOSS) projects has recently been proposed as a potential remedy to these shortcomings. Students in courses with FOSS were able to deal with existing codebases, often large and complex, such as the one they will face when working in the industry [S8b], [S8c], [S112], [S113]. Some courses also had a continuation of earlier projects in the course to expose students to code generated by other people [S14], [S15], [S19], [S38], [103]. Both of these approaches allow students to maintain existing code, but they still represent a minority in our research.

4.4.3. Project technologies (RQ4.3)

We also looked into the development technologies used in capstone courses and how they were selected (F12). Commonly, in multi-customer courses or in courses with otherwise very differing project ideas, the technology choices were made based on the project (Table 11). In these cases, the course staff did not impose a common technology stack for all the projects. For some of these projects, the technology stack was based on the client's infrastructure [S35]. In other cases, the students made manager-like decisions on the suitable development technologies [S6], [S84]. Having the teams decide on the tools and technologies made the students explore available options and justify their selections [S6], [S56], [S84]. However, even though the majority of technologies would be selected based on the project, some of the reviewed articles recommend having some shared infrastructural tools and technologies across teams [S12], [S19], [S36], [S65], [S102]. Version-control [S6], [S12], [S19], [S29], [S36], [S67], [S102], project management and communication tools [S19], [S29], [S65], [S67], [S80] and tools for continuous integration and delivery [S32], [S78] are examples of these. Having these technologies common for all the teams made the management and evaluation of projects easier [S19], [S29]. Development technologies were fully common across teams especially in cases where the teachers provided students with the project requirements [S42]. In some cases, the decision to use a common technology stack was based on the evaluation methods being heavily focused on technical implementation. For example, the course graders had sets of tests to run to determine the quality of each project [S109]. Some educators had the students competing on the same project proposal, which made choosing a common stack justifiable [S37].

4.5. Assessment of students (RQ5)

We were also interested in finding out how the assessment was conducted in these courses (F13), and to which extent students were given possibilities to reflect on their experiences. We looked at both the end-of-course student assessment (Section 4.5.1) and any continuous guidance and feedback students were given during the course (Section 4.5.2).

4.5.1. End of course student assessment (RQ5.1)

All articles which explicitly describe the course evaluation process had course teachers involved in it (Table 12). Teachers were generally the ones assessing any artefacts produced on the course such as the software product itself, reports and documentation [S118]. Several articles also report having additional sources for student assessment beyond the teachers' evaluation of produced artefacts. Reasons for having additional sources included the decreased quantity of delivered artefacts when using agile methodologies [S110]. Having a limited view of teamwork and skills development in a non-classroom course setting was also mentioned [S52], [S56], [S86]. Different kinds of anonymous peer evaluations were fairly common (31%). They gave course staff a look into the team dynamics during the course and helped in detecting free-rider behaviour [S12], [S86], [S112]. Self-evaluation was done either in combination with peer evaluations [S56] or as a part of the reflection done in the final project report [S65].

Table 12
End of course assessment.

Category	Number of courses	Percentage	Article identifiers
Course staff	93	71%	S1, S2, S4, S6, S7, S8a, S8b, S8c, S8d, S9, S12, S13, S17, S18, S19, S20, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S33, S35, S36, S37, S39, S40, S41, S42, S43, S44, S46, S47, S48, S51, S52, S53, S56, S58, S62, S63, S64, S66, S67, S68, S69, S71, S72, S73, S74, S77, S80, S81, S82, S84, S85, S86, S87, S88, S89, S90, S94, S95, S96, S97, S98, S99, S100, S101, S102, S103, S104, S105, S106, S107, S108, S109, S110, S111, S112, S116, S117, S118, S120, S123, S124, S126, S127
Students' peer-evaluations	40	31%	S2, S4, S7, S8a, S8b, S12, S13, S27, S30, S31, S33, S37, S41, S42, S46, S48, S51, S52, S56, S58, S63, S64, S67, S72, S73, S74, S81, S84, S86, S87, S90, S98, S106, S107, S108, S110, S112, S117, S124, S127
Students' self-evaluations	24	18%	S1, S2, S4, S22, S23, S27, S37, S41, S42, S47, S51, S56, S62, S64, S69, S73, S74, S80, S81, S86, S87, S116, S126, S127
External project clients	17	13%	S4, S20, S24, S29, S33, S35, S37, S42, S58, S73, S84, S85, S86, S89, S96, S98, S127
Others	1	1%	S17
Not specified	38	29%	S3, S5, S10, S11, S14, S15, S16, S21, S32, S34, S38, S45, S49, S50, S54, S55, S57, S59, S60, S61, S64, S65, S70, S75, S76, S78, S79, S83, S91, S92, S93a, S93b, S113, S114, S115, S119, S121, S122

Table 13
Continuous student assessment and guidance.

Category	Number of courses	Percentage	Article identifiers
Course staff	99	76%	S2, S3, S4, S6, S8a, S8b, S8d, S9, S12, S15, S16, S17, S18, S19, S20, S22, S23, S24, S25, S27, S28, S29, S30, S31, S32, S35, S36, S37, S39, S40, S41, S43, S44, S45, S46, S47, S48, S49, S50, S51, S52, S53, S54, S55, S56, S57, S59, S60, S62, S63, S64, S65, S66, S67, S68, S69, S71, S73, S74, S75, S76, S77, S78, S80, S81, S82, S84, S85, S86, S87, S88, S90, S91, S92, S93a, S94, S95, S96, S97, S98, S99, S100, S101, S102, S103, S105, S106, S108, S109, S110, S112, S113, S114, S115, S116, S122, S123, S124, S126, S127
More experienced students	23	18%	S5, S8a, S12, S14, S17, S19, S35, S40, S48, S58, S61, S68, S81, S85, S92, S95, S104, S105, S112, S113, S118, S119, S121
Industry advisers (other than project clients)	22	17%	S5, S8a, S8b, S8c, S10, S20, S25, S52, S58, S65, S71, S72, S73, S77, S80, S83, S89, S93b, S98, S106, S118, S120
Not specified	15	11%	S1, S7, S11, S13, S21, S26, S33, S34, S38, S42, S70, S79, S107, S111, S117

Some studies report utilising the client's opinion in the course assessment process. Clients sometimes filled out a questionnaire considering each student's performance during the course [S86] or evaluated the team's deliverables and their value from the client's point-of-view [S35], [S52], [S89], [S127]. As with self- and peer-reviews, educators used the client's opinion as a complementary source of assessment when grading students (Table 12).

4.5.2. Continuous student assessment and guidance (RQ5.2)

Many of the reviewed articles specifically mention that the teams should not be left entirely on their own to complete the course project and should be guided along the way [S6], [S9], [S10], [S16], [S18], [S53], [S69], [S72], [S86]. Three main ways for conducting such continuous assessment and guidance were found (Table 13). Articles where there was no mention of the teacher, or anyone else, having an active role in how the teams worked during the course fell into the category "Not specified". If the course staff only passively received reports of students' progress and evaluated the course outcomes after its completion, these were not the active guidance we were looking for. Some courses had several types of guidance present, in which case the article was listed under each corresponding category in Table 13. Only 11% of the reviewed articles does not explicitly specify having any ongoing feedback and guidance system present during the course.

The most common way was to have the course staff, such as the responsible teacher or hired teaching assistants, acting in an advisory role (76%). The intensity of the guidance given by course staff varied a great deal between, or even within, these courses. Sometimes course staff provided oversight in a more supervisory role and intervened in the team's work only if conflicts arose or some students clearly did not contribute to the team [S6]. In contrast, some instructors had weekly meetings with the students where the teachers actively proposed solutions and guided the teams with technical and non-technical issues and team dynamics [S6], [S88]. Some teachers even preferred to manage the team [S6]. One article reports that the most successful changes made on the course were those that allowed the course staff to take a more active role in each team [S38]. The grades of students improved, and the teams were able to complete more functionality to their the software products.

Another, often complementary, guidance form was to have industry experts occasionally participate in the course (16%). This could be seen especially relevant when the course projects were focused around a common theme for instance the gaming industry [S25], [S106]. However, finding the correct balance in this type of guidance was

sometimes tricky. One course had industry experts from the gaming and software industries participating as advisers [S106]. While the course staff took the advisers' encouraging feedback to mean that the game concept and development were merely acceptable, the students took it to mean that the incomplete prototype was, as presented, worthy of praise. This presented a dichotomy that never got resolved: students felt that the project was near-complete, whereas the instructors felt that the project was, at best, a rough sketch.

Many courses had more experienced students outside of the course staff mentoring the students in the course (18%). More experienced students were for instance students who had completed the project course themselves in the past year [S122]. This was found to benefit both the project implementation and group dynamics: an active and knowledgeable coach could, for example, help students ask clarifying questions from the customer, overcoming fear of these being stupid [S122].

Forming a capstone team of final year students with similar skill levels is in accordance with the ACM/IEEE Curriculum Guidelines for Undergraduate SE Degree Programmes [2], but leaves out an integral part of the real software development team experience: junior and senior positions. This discrepancy is noted in some of the course implementations, where senior students were involved not just as advisers but also in the actual software development [S19], [S35], [S58], [S98]. In these capstones, less-experienced students worked as junior developers and more-experienced students as senior developers or team leaders. One article presents a capstone course where students were required to work on two-course units on the same project, one unit as a junior member and one unit as a senior member [S19]. Each unit lasted one period (a quarter of an academic year), but the periods did not have to be consecutive to allow some flexibility for students in organising their studies. In order for such an arrangement to work, the projects in the course were large, long-term products which underwent enhancements over a number of semesters. Such course implementation was perceived to be a valuable and industry-relevant experience for both student groups.

5. Discussion

Firstly, we summarise the main findings of this literature review and compare them to the findings of previous systematic literature reviews whenever appropriate (Section 5.1). Secondly, we discuss the validity of the results in Section 5.2.

5.1. Main findings

5.1.1. Duration (RQ1)

Despite [2] recommending that undergraduate SE capstones should span the whole academic year, most courses in this research lasted only one semester. Average duration of one semester is in line with the findings by Dugan Jr. [21], so it appears that there has been no changes in this regard over time. In our research, the articles presenting two-semester capstones often find one-semester courses inadequate in depth and breadth of skills they can provide. According to Dugan Jr. [21], a longer course better prepares students for the experiences they can expect in their working life in software engineering. There were, however, some real-world constraints to why the courses were generally shorter, such as cramped curricula and the time and effort longer capstones require from the staff and students.

5.1.2. Team sizes (RQ2)

In our survey, most capstone courses were large-scale group projects. This aligns well with the ACM/IEEE [2] recommendations. The team sizes varied greatly between courses, ranging from 1 to 35 students in one team. While Dugan Jr. [21] found no agreement in the literature on the appropriate team size, in our research, several articles that report having experiences with different team sizes found the optimal to be 4–5 students per group. The average team size observed in this study matches the perceived optimal team size. Groups of 2–3 students did not have any communication challenges to solve, and smaller groups often could not accomplish larger projects. In contrast, larger teams often had issues with communication and coordination. Larger teams also required more effort from the teaching staff to ensure fair grading and an even distribution of work.

5.1.3. Clients and project ideas (RQ3)

In our research, 58% of the reviewed articles report having external clients for student projects (RQ3.1) and projects were based on the real needs of external stakeholders in 62% of the courses (RQ3.2). ACM/IEEE highly recommends using external clients in capstones for both undergraduate and graduate degree programmes in software engineering [2,25]. Having clients outside the immediate course staff presented more work for the teachers, but working on real projects was often rewarding for students. It had positive implications for their skills and employment after the course. Despite these recommendations and benefits, there still was a considerable number of capstone courses (42%), where the course staff acted as the client for these projects, or there were no project clients at all. In addition, there were multiple cases where the teacher provided students with the project specifications (21%) or students themselves generated project proposals (17%). In these cases, students did not get the educational experience of planning a project with an external stakeholder.

Regarding projects and clients in other literature reviews, the taxonomy used by Dugan Jr. [21] relates to project topics and not particularly project sources or clients. This makes it difficult to assess whether there have been changes in this over the years. In the future, it would be important to explore this in more detail and evaluate the consequences of different client choices similar to what Steghöfer et al. [35] have done.

5.1.4. Project implementation (RQ4)

In 97% of the reviewed courses, students were expected to deliver a software product at the end of the course (RQ4.1). This clearly aligns with the ACM/IEEE recommendations [2], which state that a capstone course should have an implementation as an end result. Additionally, students were often required to produce agile development artefacts (e.g. product and sprint backlogs), presentations, project plans and software documentation. This supports the idea of a well-rounded software development experience as ACM/IEEE [2] recommends. In our research, the role of documentation was slightly different from the role it had in the survey done by Dugan Jr. [21]. In their classification, the core written documents include project proposals, requirements documents, project plans, designs, test plans and user manuals. In contrast, we found that when courses had shifted towards more agile development approaches the number of written assignments had reduced, and the documentation was generated throughout the course rather than as detailed plans up front. This indicates that tertiary institutions have modified the course content over time according to prevailing SE methodologies.

Regarding software life-cycle phases (RQ4.2), in many cases, the students started the projects from scratch and produced a prototype or a software product that was handed off to clients or teachers at the end of the course. Therefore, the maintenance of existing software products and working with existing codebases were usually left unexperienced. Dugan Jr. [21] made similar observations and states that maintenance was frequently mentioned in the literature with little detail of its actual implementation. Maintenance thus still remains an issue that is left with little attention in SE capstone literature, despite its high relevance in the industry. Some educators had solved this by assigning students to large projects, which underwent various incremental improvements over the years. Others had students contributing to large Open Source projects, but both of these approaches still represented a minority in our study.

The reviewed articles mostly do not explicitly describe all the technologies used in the course (RQ4.3). However, in most courses technology selections were made based on the project specifications and needs. This often entailed students learning new technologies and having to justify their selections. This is in accordance with ACM/IEEE recommending a wide set of product development life-cycle activities to be adopted in capstone courses.

5.1.5. Assessment of students (RQ5)

In all articles that discussed the end-of-course student assessment (RQ5.1), the teachers were involved in determining the final grades. Concrete deliverables (produced software, plans, agile artefacts, reports) were generally graded by the teacher. These artefacts provided teachers with some understanding of how well students understood the phases of software development. ACM/IEEE [2] suggests that assessing the deliverables should not be based solely on a working system. Capstones did include various deliverables, making them comply with this recommendation well. ACM/IEEE [2] also recommends that capstone assessment should include the quality of student reflection on the experience. In this regard, the proportion of articles that mention using self- and peer-reviews in end-of-course assessment was surprisingly low. However, many courses which did not include student reviews in grading had students writing reflective reports during the course.

Continuous assessment and guidance during the course (RQ5.2) were explicitly addressed in most of the reviewed articles (89%). Our research showed that any sort of mentoring or coaching was found to be highly beneficial for students. It increased the success rate of projects and helped teachers to identify problems early on. Course staff was the one most often guiding students during the project, and several courses had hired teaching assistants for the task. Some articles mention having more experienced students advising the capstone participants. This was found to be a rewarding experience for both student groups. ACM/IEEE [2] recommends that students should be given adequate opportunities to reflect on the experience. In the reviewed courses, students had ample opportunities to gain feedback and reflect on their choices during the course.

Trevisan et al. [23] also states that assessment during the course can provide feedback to students about their current performance and by implication, what they must do to close the gap between this performance and the instructor's expectations. Despite focusing on classroom assessment specifically, they also feel that continuous assessment is valuable in a capstone setting.

5.2. Threats to validity

5.2.1. Inaccuracy and bias in selected papers for review

One of the main limitations of any review is the possible bias in the article selection process. In our research, the reviewed articles only represent the capstone courses with some aspects or outcomes worthy of publication. Therefore the study sample might be skewed towards successful, well-planned courses or easy-to-research courses. This is not a problem when mapping out the attributes of a capstone course, such as team sizes or delivered artefacts. However, the quantitative results (e.g., the portion of the courses having an external client) should be addressed with caution as certain course aspects may be more likely in real life than in SE education research publications.

Secondly, we also acknowledge that there are similar courses organised under other related disciplines, such as data science and computer engineering. We, however, knowingly chose to leave other disciplines out of the scope of this research as we wanted to provide a classification and insights specifically on software-related capstones. We followed the SLR method by Kitchenham and Charters [29] and provided a detailed description of the paper selection process to avoid ambiguity. The ACM/IEEE [2] recommendations from where we derived our research questions were also provided specifically for software engineering capstones.

5.2.2. Inaccuracy and bias in data extraction

It is worth noting that the articles presented in this review are not exclusively written to provide course descriptions or general course evaluations. Some articles have a section dedicated to the course overview, which might have provided all the details of the course structure we needed. Then again, some articles had the relevant details scattered across various sections and might not have been explicitly referred to as our categories suggest. Especially regarding the produced artefacts and student assessment, the descriptions varied greatly in terms of detail and clarity. Further, the first author did the data extraction and quality assessment without systematically cross-checking analyses with other authors, which may have caused some bias.

To mitigate these problems, we tried to keep the data categories generic and descriptive, so that it would be easy to grasp the general outline of each course. We also refrained from reading too much into the text itself and created the category "not specified" for unclear cases. Regarding quality assessment, the two summed-up categories presented, rigour and credibility, aimed to diminish the impact of a single question and evaluate the article rather as a whole.

6. Conclusions

This research aimed to understand how software engineering capstone courses are organised in tertiary education institutions. For this purpose, we conducted a systematic literature review, including 127 articles on real-life SE capstone courses. The characteristics were synthesised into a taxonomy consisting of duration, team sizes, clients and project sources, project implementation and student assessment. Based on the synthesised justifications and outcomes for these characteristics, we provided an overview of how the courses can be organised and what the trade-offs are to be weighted regarding each characteristic.

Summarised takeaways of the characteristics identified from articles are as follows:

- Most capstone courses lasted one semester.
- Capstones were almost always conducted as team-based project courses.
- The most common team size was between 4–5 students. This was found to be optimal in many of the reviewed articles.
- Course projects were often lacking an external client (42%).
- Nearly all courses (97%) expected a software implementation as the main deliverable.
- Number of artefacts produced during the course had diminished over time as courses had moved to use agile development methodologies.
- Teachers graded the produced artefacts and were always involved in the end-of-course assessment.
- Students' self- and peer-reviews and clients' opinions were used in grading in a minority of courses.
- Nearly all (89%) articles reported that their course had regular guidance available for students during the projects.

We found that for significant parts the courses were aligned with the recommendations given by ACM/IEEE [2] for software engineering capstones. The courses had a software implementation as the main deliverable, the students were assessed based on various factors, not just the delivery of a working system, and the projects in these courses were almost always completed as group assignments. Students were also often given guidance and continuous assessment throughout the course via written and oral feedback on their progress and deliverables. Although many of the courses follow the guidelines, there are differences. Moreover, our taxonomy of the course features illustrates how capstone courses can be implemented in many ways. Thus, studies related to capstone courses should carefully describe the contextual details, e.g., by using our framework.

The areas with the most discrepancies to ACM/IEEE [2] recommendations were the duration of the course and the use of external clients. The courses generally lasted one semester, while ACM/IEEE recommends two semesters. A considerable number of courses did not have a client external to the course staff, despite external clients being recommended for undergraduate and graduate capstones [2,25]. Also, only a minority of courses covered maintenance; the projects usually progressed from idea to a basic product or prototype handed over at the end of the course.

Especially regarding these discrepancies, it would be worthwhile to see more experiments where the presented characteristics would be controlled in order to understand their individual roles in the course outcomes. Moreover, if the reality and guidelines are not aligned, it would be important to understand which one should be changed. Most of the research identified here does not provide controlled, comparative results on the capstone characteristics.

Usually, at least one of the authors of the study was somehow involved in organising the course in question. Additionally, quite a large portion of these reports lacked an honest evaluation of the author bias, as can be seen in Section 3.3.3. Therefore there is an inherent lack of truly objective third-party assessment of SE capstone courses in the literature. We would welcome more research on capstone courses, or on SE education in general, where the author is an unbiased third party. As capstones are meant to train students for the work-life, we would also like to see more research done on how well these courses capture what students face later on in their careers.

CRediT authorship contribution statement

Saara Tenhunen: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualisation. **Tomi Männistö:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Matti Luukkainen:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Petri Ihantola:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2023.107191.

Data availability

Data will be made available on request.

References

[1] ACM/IEEE, ACM/IEEE joint task force on computing curricula: Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science, 2013, URL https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf (visited on 4/20/2022).

[2] ACM/IEEE, ACM/IEEE joint task force on computing curricula: Software engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering, 2014, URL https://ieeecs-media.computer.org/assets/pdf/se2014.pdf.

[3] A. Radermacher, G. Walia, D. Knudson, Investigating the skill gap between graduating students and industry expectations, in: Companion Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 291–300.

[4] V. Garousi, G. Giray, E. Tuzun, C. Catal, M. Felderer, Closing the gap between software engineering education and industrial needs, IEEE Softw. 37 (2) (2019) 68–77.

[5] M. Ikonen, J. Kurhila, Discovering high-impact success factors in capstone software projects, in: Proceedings of the 10th ACM Conference on SIG-Information Technology Education, 2009, pp. 235–244.

[6] H. Ziv, S. Patil, Capstone project: From software engineering to "Informatics", in: 2010 23rd IEEE Conference on Software Engineering Education and Training, IEEE, 2010, pp. 185–188.

[7] A.-M. Majanoja, T. Vasankari, Reflections on teaching software engineering capstone course, in: CSEDU (2), 2018, pp. 68–77.

[8] R.C. Panicker, S. Sasidhar, S.Y. Jien, C.K.-Y. Tan, Exposing students to a state-of-the-art problem through a capstone project, in: 2020 IEEE Frontiers in Education Conference, FIE, IEEE, 2020, pp. 1–8.

[9] K. Keogh, L. Sterling, A.T. Venables, A scalable and portable structure or conducting successful year-long undergraduate software team projects, J. Inf. Technol. Educ.: Res. 6 (1) (2007) 515–540.

[10] E. Venson, R. Figueiredo, W. Silva, L.C. Ribeiro, Academy-industry collaboration and the effects of the involvement of undergraduate students in real world activities, in: 2016 IEEE Frontiers in Education Conference, FIE, IEEE, 2016, pp. 1–8.

[11] K.Z. Watkins, T. Barnes, Competitive and agile software engineering education, in: Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon), IEEE, 2010, pp. 111–114.

[12] R. Dupuis, R. Champagne, A. April, N. Séguin, Experiments with adding to the experience that can be acquired from software courses, in: 2010 Seventh International Conference on the Quality of Information and Communications Technology, IEEE, 2010, pp. 1–6.

[13] H.M. Haddad, One-semester CS capstone: A 40-60 teaching approach, in: 2013 10th International Conference on Information Technology: New Generations, IEEE, 2013, pp. 97–102.

[14] J. Bowring, Q. Burke, Shaping software engineering curricula using open source communities, J. Interact. Learn. Res. 27 (1) (2016) 5–26.

[15] M. Paasivaara, J. Vanhanen, C. Lassenius, Collaborating with industrial customers in a capstone project course: the customers' perspective, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), IEEE, 2019, pp. 12–22.

[16] K.P. Anicic, Z. Stapic, Teaching methods in software engineering: Systematic review, IEEE Softw. (2022).

[17] M.R. Marques, A. Quispe, S.F. Ochoa, A systematic mapping study on practical approaches to teaching software engineering, in: 2014 IEEE Frontiers in Education Conference (FIE) Proceedings, IEEE, 2014, pp. 1–8.

[18] O. Cico, L. Jaccheri, Industry trends in software engineering education: a systematic mapping study, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2019, pp. 292–293.

[19] O. Cico, L. Jaccheri, A. Nguyen-Duc, H. Zhang, Exploring the intersection between software industry and software engineering education-a systematic mapping of software engineering trends, J. Syst. Softw. 172 (2021) 110736.

[20] L.L. Fortaleza, T. Conte, S. Marczak, R. Prikladnicki, Towards a GSE international teaching network: Mapping global software engineering courses, in: 2012 Second International Workshop on Collaborative Teaching of Globally Distributed Software Development, CTGDSD, IEEE, 2012, pp. 1–5.

[21] R.F. Dugan Jr., A survey of computer science capstone course literature, Comput. Sci. Educ. 21 (3) (2011) 201–267.

[22] N. Martin, Designing the IT capstone course: A systematic literature review, in: Proceedings of the 20th Annual SIG Conference on Information Technology Education, 2019, pp. 102–102.

[23] M. Trevisan, D. Davis, S. Beyerlein, P. Thompson, O. Harrison, A review of literature on assessment practices in capstone engineering design courses: Implications for formative assessment, in: 2006 Annual Conference & Exposition, 2006, pp. 11–112.

[24] J.E. Burge, G.C. Gannod, Dimensions for categorizing capstone projects, in: 2009 22nd Conference on Software Engineering Education and Training, IEEE, 2009, pp. 166–173.

[25] ACM, Graduate software engineering 2009(gswe2009) curriculum guidelines for graduate degree programs in software engineering, 2009, URL https://www.acm.org/binaries/content/assets/education/gsew2009.pdf.

[26] R.L. Glass, V. Ramesh, I. Vessey, An analysis of research in computing disciplines, Commun. ACM 47 (6) (2004) 89–94.

[27] R. Parker, S. Sangelkar, M. Swenson, J.D. Ford, Launching for success: A review of team formation for capstone design, Int. J. Eng. Educ. 35 (6) (2019) 1926–1936.

[28] J. Hattie, H. Timperley, The power of feedback, Rev. Educ. Res. 77 (1) (2007) 81–112.

[29] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical Report, Ver. 2.3 EBSE Technical Report EBSE, Citeseer, 2007.

[30] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Inf. Softw. Technol. 50 (9–10) (2008) 833–859.

[31] M.S. Ali, M.A. Babar, L. Chen, K.-J. Stol, A systematic review of comparative evidence of aspect-oriented programming, Inf. Softw. Technol. 52 (9) (2010) 871–887.

[32] S. Mahdavi-Hezavehi, M. Galster, P. Avgeriou, Variability in quality attributes of service-based software systems: A systematic literature review, Inf. Softw. Technol. 55 (2) (2013) 320–343.

[33] A. Castleberry, A. Nolen, Thematic analysis of qualitative research data: Is it as easy as it sounds? Curr. Pharm. Teach. Learn. 10 (6) (2018) 807–815.
[34] A. Mishra, D. Dubey, A comparative study of different software development life cycle models in different scenarios, Int. J. Adv. Res. Comput. Sci. Manag. Stud. 1 (5) (2013).
[35] J.-P. Steghöfer, H. Burden, R. Hebig, G. Calikli, R. Feldt, I. Hammouda, J. Horkoff, E. Knauss, G. Liebel, Involving external stakeholders in project courses, ACM Trans. Comput. Educ. (TOCE) 18 (2) (2018) 1–32.

Saara Tenhunen has recently graduated with a Master of Science from the University of Helsinki (UH) and works in software development. Her interests lie in software engineering and improving software engineering education for students like herself.

Tomi Männistö received his PhD from the Helsinki University of Technology, nowadays known as Aalto University. He is a full professor at the University of Helsinki of the empirical software engineering research group. His research interests include software architectures, variability modelling and management, configuration knowledge, and requirements engineering.

Matti Luukkainen received his PhD from the University of Helsinki. He is a university lecturer at the University of Helsinki, specialising in teaching web development and software engineering. His current research interest is in computer science education.

Petri Ihantola works as an associate professor of big data learning analytics and director of the MOOC-center at the University of Helsinki, Finland. He received his PhD from Aalto University in 2011. His research interests span educational data mining and building educational software with a particular focus on smart content, automated assessment, and learning analytics in computing education.