A module-agnostic reference software development process for different levels of higher-education study

Carlos da Silva
Department of Computing
Sheffield Hallam University
Sheffield, UK
c.dasilva@shu.ac.uk

Jack Carey
Department of Computing
Sheffield Hallam University
Sheffield, UK
jack.carey@student.shu.ac.uk

ABSTRACT

Several software development methodologies and practices are taught in computer science and software engineering higher education degrees. This happens through individual modules and capstone projects and sometimes with participation from real clients. Different from industry, where processes are usually prescribed, students encounter company-agnostic artefacts and practises, often having to choose between the available options. Feedback from tutors, students and clients indicates this to be a challenge, with students often confused trying to mix-and-match different practices without the proper consideration of how they would work together. This paper introduces the SHU Development Process, covering all software development stages, that is instantiated into different levels of detail for students as they progress through their degree studies. The SHU Dev Process provides structured guidance to software development practices that can be followed through their chosen process flow or cherry-picked by students as needed. It has been created through a student-led project over multiple years. First applied during the academic year 2021/2 and iterated upon for 2022/3 in a capstone project module at Sheffield Hallam University, the process was evaluated annually by surveying students across different courses and levels of study. In initial surveys, students responded positively, and our experience provides valuable insight that other practitioners may draw upon to implement and evaluate a similar resource in the future.

CCS CONCEPTS

• **Software and its engineering** → **Software development methods**; **Collaboration in software development**; • **Social and professional topics** → **Computing education**; **Software engineering education**; **Computer science education**.

KEYWORDS

software engineering education, software development process, higher education, students, university

UKICER 2023, September 07–08, 2023, Swansea, Wales Uk
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0876-3/23/09…$15.00
https://doi.org/10.1145/3610969.3611122

ACM Reference Format:
Carlos da Silva and Jack Carey. 2023. A module-agnostic reference software development process for different levels of higher-education study. In *The United Kingdom and Ireland Computing Education Research (UKICER) conference (UKICER 2023), September 07–08, 2023, Swansea, Wales Uk*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3610969.3611122

1 INTRODUCTION

It is common practice in software development organisations to define some kind of methodology that documents and provides guidance to software development teams [8, 10]. Examples of such guidance include different variations of agile [3] practices and more structured approaches such as Capability Maturity Model Integration (CMMI) [5]. More recently, we can also see strategies for structuring large teams, such as the Spotify Tribes Model which divides a large team into squads and tribes [11] and its use for different scenarios [17].

During computer science and software engineering degrees, students are exposed to different concepts related to software processes and methodologies, through a number of techniques. These often come into play during capstone projects, in which students are required to build a relatively complex piece of software combining knowledge and skills learned from previous modules, and sometimes associated with a real client [14]. It is known that effective student outcomes can improve by as much as 57% by being involved with industrial partners [14], especially regarding software development aspects beyond coding, such as communication, version control, documentation, and planning.

The Computer Science & Software Engineering (CS&SE) subject group of Sheffield Hallam University (SHU) adopts the use of capstone projects for first and second year students. Delivered during the second semester of teaching, they combine aspects of problem-based learning and interaction with real clients. In both modules, student teams communicate with clients to gather requirements and use these to plan their approach to the task at hand, including aspects related to needed development infrastructure, technologies and initial design. This culminates in an intense sprint week with a mixed vibe of a hackathon and office-hours where students build their respective solutions and present it to the client at the end of the week.

After running these modules for many years, the teaching team noticed a few problems, usually substantiated by feedback from students and clients. One of the problems noticed was a need for some structure in the software development process for students to follow. Throughout the CS and SE degrees students are exposed to different methodologies, methods and techniques involved in software development projects, being free to use them as they choose. However, with no standardised reference to be followed, we started to notice some discrepancy in the produced output of capstone projects. Taking the capstone module as example, we noticed that those teams that got involved with clients that work with software development were absorbing aspects of the client's software development practices into their own projects, while those with clients in other domains would adopt a more ad-hoc approach, usually getting overwhelmed by the sheer amount of practices available. This was seen in the 2021/22 academic year with a team that worked with a physiotherapist for tracking the progress of their patient exercises. As the client was not involved with software engineering, the group did not have a point of reference to guide their development work.

In this context, and incited by some students, we defined a reference development process that can be used by students to organise their work in capstone projects. This paper presents the *SHU Development Process* (SDP)[1] a module and course agnostic software development process that can be instantiated into different levels of teaching. The SHU Development Process provides a structured reference for student projects, with a standard set of artefacts for students to use when they engaged with software development projects, write reports, give presentations, and demonstrate their work. The SDP has been officially incorporated into our second year (level 5) capstone project module during the academic year 2022/23. Finally, the paper also presents an evaluation of the SDP which involved a survey with the cohort of academic year 2021/22 and 2022/23 of Department of Computing at SHU. The results obtained provided valuable insights on the SDP, with some lessons learned and suggestions of improvement that will be carried out during the academic year 2023/24.

This paper is organised as follows: Section 2 presents some related work. Section 3 details the SHU Development Process. Section 4 presents our survey results together with a discussion on the lessons learned. Section 5 concludes the paper and points to future work.

2 RELATED WORK

This section presents a brief discussion on related work in the topics of problem-based learning and capstone projects, followed by a discussion on why we chose the reference documentation format instead of a question-answer based learning.

Previous research has highlighted the benefits of using problem-based learning (PBL) [2] in software engineering (SE) education, with respect to learning the software development lifecycle [14], improved academic results [16], exposure to different methods [4], and enthusiasm [6]. Paasivaara et. al. [14] showed that university modules with external clients can be the first time students understand the whole life cycle of a software development project and that the enforcement of scrum based processes limited student satisfaction. A systematic mapping of trends in SE education highlighted the need for tools that facilitate the adoption of current practices together with mixed models and methods in an educational context [4]. These findings corroborate the SHU approach of using capstone-project based on PBL and involvement of real clients together with the reference provided by the SHU Development Process documentation.

In their research on project-based learning, Włodarski et al. [21] focused on using a combination of sequential and iterative approaches. They found that processes should be adapted to academic settings and hands-on experience with all stages of the software development process [15] can bridge the gap between SE education and industry. As both sequential and iterative approaches are covered at SHU, the SHU Development Process does not prescribe one approach, instead acting as a "scaffolding" to aid the approach taken by students.

Scaffidi [18] showed that, in addition to the technical skills related to solution domains, employers are looking for skills related to methods and practises such as agile, code management, and system interface design. They showed that employers also look for soft skills relating to communication and handling ambiguous requirements, but that the value of any one skill varies between employers. The sample size of this study was small, at only 11 employers, but supports the delivery of a variety of skills in computer science and software engineering courses. Along those lines, Cico et. al. [4] showed that "*the actual participation of industrial stakeholders in SE education remains limited*", with a request for more experience papers. A literature review by Garousi et. al. [7] supports Scaffidi's finding in their examination of 33 papers. They found an importance of soft skills, as well as SE models and methods, SE process, design and architecture, and testing. These are taught at SHU within modules as well as through project-based learning in the capstone project modules at levels 4 and 5.

One interesting point considered when choosing the format of the SHU Dev Process is about the use of question & answer based learning and its benefits for student outcomes [9, 12, 20]. By interacting with teaching staff or their peers, students were able to improve exam results, as well as inform their own learning and that of their peers. While it is possible to build a knowledge repository using digital tools (for example, stack overflow), the effectiveness of the platforms can be compromised by similar questions being asked multiple times, lack of (continued) student involvement, and the need for moderators or lecturers to stay up to date with the questions being asked. For these reasons, the SHU Development Process has been designed as reference documentation instead. It is designed to be used alongside the variety of in-classroom approaches that teaching staff adopt, which range from direct instruction to question-and-answer based sessions or constructivistic seminars.

Another influence to the SDP documentation is the notion of process fragments. Seidita et al. [19] defined a process fragment as "*a portion of design process adequately created and structured for being reused during the composition and enactment of new design processes both in the field of agent oriented software engineering and in other ones.*". This has been combined with the notion of scaffolding [1] in which the different fragments are the artefacts such as user stories and use case diagrams that students develop collaboratively and use as building blocks when implementing their software.

3 THE SHU DEVELOPMENT PROCESS

The SHU Dev Process (SDP) was born based on the need for a structured reference that can be followed by students working on

[1]https://aserg.codeberg.page/shu-dev-process

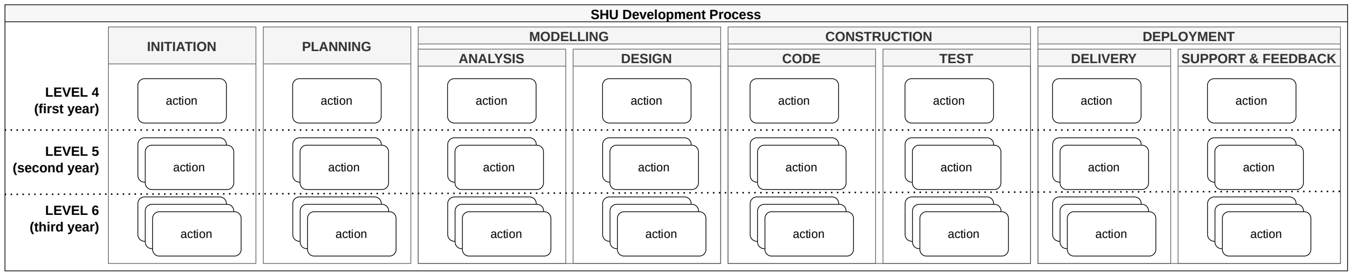

Figure 1: General view of the SHU Dev Process structure identifying its five main activities, where each block represents an action that can have greater levels of detail.

more complex software development projects. The aim is to design a software development reference that can be "instantiated" into the different levels of computer science and software engineering (CS&SE) courses. Students at SHU begin undergraduate study at level 4 (first year) and finish their third year at level 6. Our proposal structures and standardises key activities that students undertake when doing their software development coursework, including simple project management guidance that works for both individuals and teams. Where needed, complexity is added to a simple process according to level of study, in which relevant tools and techniques are detailed for students so they may better understand and apply them as they progress through their degree.

The initial iteration of the documentation was created by two "student researchers" (students hired part-time by the University to work on projects defined by academics) under the supervision of teaching staff, where the majority of the decisions related to detailing of the SDP was performed by the student themselves. The SDP documentation is designed to be consumed by any student in the University that is involved in aspects of software development projects, however attention was initially given to students on the CS&SE degrees due to its relevance to their assignments. Specifically, we targeted second year undergraduates (level 5) students undertaking a capstone project module in which external industry partners provide briefs and act as clients for student teams to develop a complex software system.

This section gives some details on the SDP followed by a discussion on its use through different navigation styles.

3.1 Detailing the SHU Dev Process

The SHU Dev Process, which can be seen in Figure 1, follows the definition by Pressman & Maxim [15] and comprises five methodological activities: initiation[2], planning, modelling, construction and deployment. These main activities encompass a set of support steps: analysis and design related to modelling; code and test related to construction; and delivery, support and feedback for deployment. These stages are used to group the varied actions involved, which are then presented with different levels of detail depending on the level of study. As one example we consider user stories which are explored throughout multiple levels of study, but with different expectations in terms of its complexity:

- **Level 4** - Introduce the *"As a [type of user] I want to [perform task] so that I can [achieve goal]"* way of thinking about functionality.
- **Level 5** - Builds on level 4 to add acceptance criteria in the format of *"Given [some context], when [an action is carried out], then [outputs occur]"* to help students scope, communicate, plan, and test functionality.
- **Level 6** - Builds on level 5 to add requirements traceability matrices to ensure the functionality agreed through user stories has been implemented as expected.

In the sequence we detail each of the main activities, describing the main topics included in the SDP according to Pressman & Maxim [15].

The **initiation** activity happens "*before any technical work can commence, and helps in establishing the communication and collaboration between the development team and the customer (and other stakeholders). The intent is to understand their objectives for the project and to gather requirements that help define software features and functions*". In particular, the SDP includes material on helping students understand the client's objectives, such as the use of socratic questioning method to interrogate the client brief and guide these initial meetings, as well as providing guidance on how to take meeting minutes.

A software project is a complicated journey, and "*the **planning** activity creates a "map" that helps guide the team through it. The map describes the software engineering work with the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule*". The SDP contains material on helping students planning for their projects, from establishment of team communication channels, the definition of SMART goals, and different aspects related to project management, such as, agile methodologies, scrum, risk management, version control workflows and respective tools.

The **modelling** activity concerns the creation of different models used in a software development project. "*Models and "sketches" are useful in software engineering, as with many other industries, to understand the big picture for projects - what they will look like architecturally, how the constituent parts fit together, and many other design characteristics. They may be redefined into greater detail in an effort to better understand the problem you're trying to solve*". Modelling is divided into analysis and design. The *analysis* is dedicated to the formalisation of the requirements. It involves talking with a client and documenting your understanding. The SDP describes

[2]We re-labelled this from the original 'Communication' section to reduce ambiguity between inter-stakeholder and intra-team communications.

different levels of complexity for user stories, use cases, personas and user scenarios, and the MoSCoW prioritisation method[3]. The *design* is focused on the modelling of the solution. It usually involves decision around architecture, data structure, user-interface, classes, algorithms, etc. The SDP includes material on class, entity-relationship and sequence diagrams with different levels of complexity based on the level of study. We also include discussions on architectural overview using the C4 model and architectural decision records.

The **construction** activity involves the implementation and testing of the structures modelled in the design phase. This activity is divided into code and test. The *code* step concerns the production of source code, with the SDP covering topics related to good practices with references to coding standard to different programming languages and discussions on commenting and review strategies. The *test* step deals with general strategies for performing software testing, focusing on unit, integration and user acceptance testing with pointers to tools and libraries on different programming languages. We also include references to code coverage.

The **deployment** activity deals with the "conclusion" of the development project. It captures the notion that "*the software is delivered to the customer who provides feedback based on their own evaluation of the delivered product*" and receives support (e.g., maintenance) from the development team, usually driven by some sort of monitoring of the running software. In the SDP, the *delivery* presents guidance on producing the artefact that will be submitted as the student's coursework. It involves creation of structured readme files with building/deployment instructions, to the full automation delivery pipeline, establishing the link with DevOps practices and tools, and including references to contemporary technologies such as containers and cloud computing services. The *support and feedback* is focused on the student, in particular, with guidance on the production of reflective statements and project reports. We also cover guidance on presentations, from general structure and points to cover, to practical tips in preparing for live demonstrations of the running software.

3.2 Navigating the SHU Development Process

One aspect that has been substantially improved in the SDP is related to its navigation. Although we use Pressman and Maxim [15] general framework to structure the material, we do not prescribe any particular order in which actions should be performed. In fact, we follow the authors' descriptions of process flow types for how activities occur with respect to sequence and time: linear, iterative, evolutionary, and parallel. The SDP adds 'ad-hoc' as an exception, for short assignments that do not assess students on all software development activities.

- **Linear** *executes each of the five activities in sequence, beginning with initiation, and culminating with deployment.*
- **Iterative** *repeats one or more of the activities before proceeding to the next.*
- **Evolutionary** *executes all five activities in a "circular" manner, with each iteration leading to a more complete version of the software.*
- **Parallel** *executes one or more activities in parallel, e.g. modelling for one aspect may take place at the same time as construction of another.*
- **Ad-hoc** execute single activities on a one-off, exceptional basis.

[3]MoSCoW and other specific techniques have been included as requests from teaching staff, but other strategies are also mentioned and linked to.

Through specific module work, students are taught to consider prescriptive methods such as waterfall, prototype, spiral or other, and how they may or may not be applicable in a given context. They can do this purely at their level of study, or above or below for increased and decreased levels of detail in specific practices. In order to make this more explicit, the structure of the SDP and the viewing controls in place allow students to navigate between artefacts using their chosen process flow without prescribing any specific one.

Figure 2 gives examples of two possible paths students may choose to apply within their projects and coursework. The "red" flow captures a student navigating the process in an evolutionary style, in which each iteration increases the level of detail as the student becomes familiar with the activites involved. In contrast, the "blue" flow demonstrates one possible path of an iterative flow, combining different levels of details where students work largely at their level of study (level 5), but increase the level of detail in their modelling as the work progresses. It is important to mention that these two are just examples of possible navigation paths and that other flow types have been observed by teaching staff throughout the second year capstone project module in the academic year 2022/23, with ad-hoc navigation being most popular for individual students.

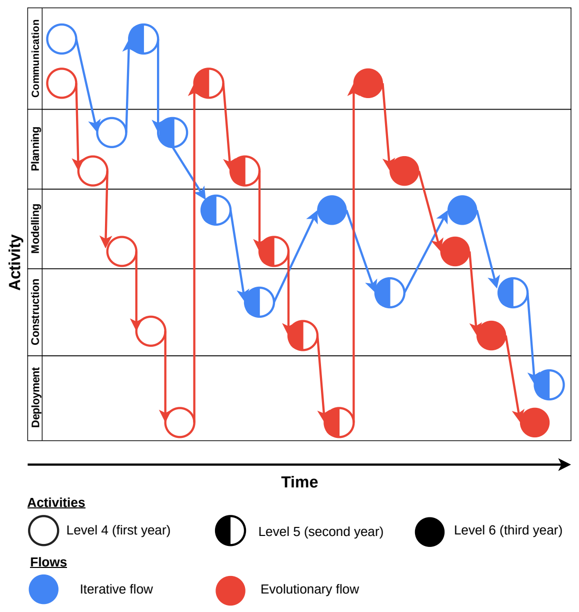

Figure 2: Two examples of possible navigation paths showing iterative and evolutionary flows.

4 EVALUATION

This section presents the results of a survey conducted during academic years 2021/22 and 2022/23, followed by a brief discussion of the obtained results.

4.1 Survey Results

During academic year 2021/22 the SDP has been made available to students of the second year capstone project module from the CS&SE degrees as one of the resources available to them. At the end of the academic year we invited these students to answer a survey with their impression of the SHU Dev Process. These provided feedback that has been incorporated into the SDP for 2022/23, in which the SDP was again indicated as online resource, and another survey performed. The surveys have been registered and approved by University ethical procedures and were completely anonymised at the point of collection, consisting of a combination of likert scales and open comments, where all questions were optional. Thematic analysis was used to categorize the open comments and a summary of each theme has been placed with each.

Table 1: Number of students participating in the survey according with their course.

	Responses / Population	
Course	**2021/22**	**2022/23**
Computer Science	17 / 186	14 / 204
Software Engineering	17 / 122	7 / 164
Other	12 / –	12 / –

We start by presenting a brief reflection on the survey population. Table 1 presents the number of students that answered the survey followed by the available population for the degrees of computer science and software engineering (our main target demographics) for both 2021/22 and 2022/23. 74% of responses (from a total of 46) came from our target demographic of computer science (CS) and software engineering (SE) courses in 2021/22, with 66% (from a total of 35) in 2022/23. The "other" courses include game design (GD), cyber security (CSec), IT management and others that are offered by different subject groups, and thus not officially involved with the SHU Dev Process. It was particularly surprising to see CSec students (5 responses out of 12) interested in the SDP documentation. We believe their participation in the survey was motivated by word-of-mouth between students, which gives us the chance to get some insights about the perspective of being module and course agnostic.

After capturing demographic information the survey continues with an open-ended question: "*What are you looking for in a resource like this?*". We received a total of 24 responses in 2021/22 (and 23 responses in 2022/23), which have been collated using thematic analysis. The main theme in both years relates to some sort of guidance on different aspects of a software development project, from concepts and project structure to coding and specific technologies. A second recurrent theme is related to feedback and interaction with other students, in particular in 2022/23 there were several mentions of inclusion of student's suggestions in the SDP itself. In both years we find responses related to summary of concepts and, in 2021/22, one explicit mention to accessibility to all reading levels. Other themes that appeared in 2022/23 revolve around help for securing jobs and personal/professional development. Topics considered out of scope for this project, such as those mentioning wellbeing or course/module-specific guidance, highlighted the need to refer students to module content and other University services in these cases. We also had a number of comments classified as "non-answer", including comments like *"not sure"*, or *"not much"*.

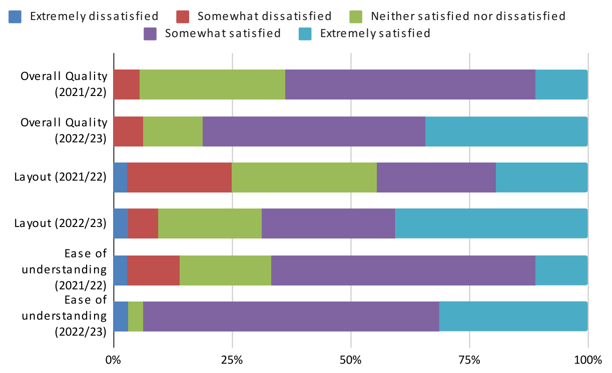

Figure 3: Results on levels of satisfaction of students with the SHU Dev Process using likert-scale.

Figure 3 presents the levels of satisfaction results obtained using likert-scale. These consider the overall quality of the documentation, the layout and ease of understanding. We draw attention to improvements in all three topics, particularly in the layout and ease of understanding, as these were tackled in response to feedback collected in 2021/22.

We also asked "*Does the documentation enhance your understanding of the software development process?*". The overall response is around 77% yes in 2021/22 (and 85% in 2022/23), with the majority of students in the target CS (85% in 2021/22 and 93% in 2022/23) and SE (90% in 2021/22 and 86% in 2022/23) courses felt the documentation enhanced their understanding of software development. We were surprised by the positive results from CSec (100% yes in both years) and negative result from GD students (20% yes in 2021/22 and 0% yes in 2022/23). CSec students do not engage in software development during their degree, possibly making the SDP the first learning material devoted to software development guidance they have been exposed to. On the other hand, GD students, as part of their degree, build a running game for a commercial platform which, as indicated by course teaching staff, involves several practices related to software development.

The survey continues by asking students "*What do you think could change to make the Reference Development Process better?*". The thematic analysis of the responses for both years identified content as the most mentioned topic, including addition of more specific details and examples, and more conciseness in some parts. The responses for 2021/22 included several suggestions for layout and accessibility, which have been incorporated into the SDP and mentioned as positive aspects in 2022/23. Both surveys identified the definition of a single example project to be used throughout the whole SDP. A consistent example project to use in diagrams and explanations is being considered for the next iteration of the SHU Dev Process, with care being taken to avoid technology-specific solutions. Another recurrent topic in both years is formal presentation of the SDP for first year students, which will be officially included in the academic year 2023/24.

4.2 Discussion

Overall, the feedback we collected supported the aims of providing a structured reference process for students. The participation of students from other courses was a surprise to us, but provided interesting comments and helped us in demonstrating the course and module-agnostic nature of the SDP. Together, these comments support the introduction of the SDP in earlier levels of study.

The SDP has also been used as inspiration by some module teaching staff in their teaching plans and is actively being consulted for the restructuring of some modules that is currently happening at SHU. We have also seen the SHU Dev Process starting to influence assignment descriptions, although not yet formally used to structure the marking of student projects. Further discussions with the teaching team are needed, particularly with the amount of details indicated according with the level of study. In particular, its use in the level 5 Professional Software Projects (PSP) module was well received by tutors, who noticed its influence in the students-produced artefacts. When asked about it the module leader said:

> "*The PSP module sees students working with real world clients, on real world projects. The students were introduced to the SHU Development Process and encouraged to apply it to their project work. Techniques such as MoSCoW prioritisation method helped students structure their work, in what for many would have been their first exposure to real projects.*"

The CS and SE courses passed through a restructuring for the academic year 2022/23 following a general "semesterasition" being conducted by the University and introducing a few other changes. As such, a direct comparison of grades could not be made against previous years. Future iterations of the module will continue to reference the SDP so that its use can be better evaluated. In particular, we have to consider a trade-off between the module assessment following the SDP structure too closely while maintaining its course/module-agnostic nature. One positive outcome is that the SDP has now been presented to all teachers of CS and SE degrees, with the aim of it being officially incorporated into all modules for the academic year 2023/24.

The main lesson learned with this experience is related to the involvement of "student researchers" [13]. They were involved in the definition of the process, but tutors were the ones presenting it to students as part of the modules sessions. The student researcher was able to spread the SDP through word-of-mouth but this restricts the reach to its network of students. As demonstrated by some of the received comments, students are more receptive when they could see that their peers were involved in presenting such an approach. We intend to continue involving student researchers in further development of the SHU Dev Process, as well as in presenting the SDP to peers in their respective modules.

One limitation of this experience was the sample size of respondents. They represented a small fraction of the hundreds of students in the department (as shown in Table 1), particularly when considering our targeted courses. This means that the results may not be representative of the entire computing department student body. Improvement and repetition of the survey in the next academic year or at another institution could be used to validate or disprove the results seen in this paper. Nevertheless, the survey provided areas of improvement for future iterations. The suggestions and improvements were organised as issues on the git repository[4] so they can be tracked, discussed, and implemented.

5 CONCLUSIONS

This paper presented the *SHU Development Process*, a structured reference documentation for students to follow as they learn about and implement software development practises, both within a successful project-based module and beyond. The SDP aimed to supplement the full variety of teaching approaches taken by educators by providing a canonically correct source of information that students can refer to when deciding which methods to apply in their work. We recommend that academics and educators teaching software development processes introduce similar documentation to their courses to support the effective student outcomes in their institution.

The evaluation showed that this has been beneficial to both student experience and that of the teaching staff who assess them. The results obtained through a survey with students indicates that it is worth instantiating the SDP into first year modules, which we plan to do with the capstone-project module recently introduced to level 4 of CS and SE degrees. We also intent on applying and surveying students throughout their full 3-year degree journey. The changes recommended by students and highlighted by our evaluation have been registered as issues in our git repository and will be tackled in the next academic year. Besides that, we also intend to conduct a more thorough evaluation of the SDP documentation, initially in the context of the PSP module, but expanding to other modules and courses.

The survey responses we collected focused on the affective outcomes of students using the documentation within their courses. While results supported the introduction of the documentation, no industry link was made directly between the produced documentation and current software engineering industry trends. Though the taught content at SHU is already informed by industry, further work should directly survey external partners involved with project-based modules to find gaps and/or corroboration between the produced documentation and industry practises.

ACKNOWLEDGMENTS

This project has been funded by the Teaching, Learning & Student Experience Enhancement Projects and SETL Evaluation Bursaries 2022-2023 of the College of Business, Technology and Engineering of Sheffield Hallam University.

ACKNOWLEDGMENTS

[4]https://codeberg.org/aserg/shu-dev-process

REFERENCES

[1] Sylvia Ashton and Rachel Stone. 2021. *An A-Z of creative teaching in higher education* (2nd edition. ed.). SAGE, London.

[2] Terry. Barrett. 2011. *New approaches to problem-based learning : revitalising your practice in higher education.* Routledge, New York.

[3] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for Agile Software Development. http://www.agilemanifesto.org/

[4] Orges Cico, Letizia Jaccheri, Anh Nguyen-Duc, and He Zhang. 2020. Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *Journal of Systems and Software* 172 (Feb. 2020), 110736. https://doi.org/10.1016/j.jss.2020.110736

[5] Radu Constantinescu and Ioan Mihnea Iacob. 2007. Capability maturity model integration. *Journal of Applied Quantitative Methods* 2, 1 (2007), 31–37.

[6] Maria Lydia Fioravanti, Bruno Sena, Leo Natan Paschoal, Laíza R. Silva, Ana P. Allian, Elisa Y. Nakagawa, Simone R.S. Souza, Seiji Isotani, and Ellen F. Barbosa. 2018. Integrating Project Based Learning and Project Management for Software Engineering Teaching: An Experience Report. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 806–811. https://doi.org/10.1145/3159450.3159599

[7] Vahid Garousi, Gorkem Giray, Eray Tuzun, Cagatay Catal, and Michael Felderer. 2020. Closing the Gap Between Software Engineering Education and Industrial Needs. *IEEE Software* 37, 2 (March 2020), 68–77. https://doi.org/10.1109/MS.2018.2880823 Conference Name: IEEE Software.

[8] Johannes Holvitie, Sherlock A. Licorish, Rodrigo O. Spínola, Sami Hyrynsalmi, Stephen G. MacDonell, Thiago S. Mendes, Jim Buchan, and Ville Leppänen. 2018. Technical debt and agile software development practices and processes: An industry practitioner survey. *Information and Software Technology* 96 (April 2018), 141–160. https://doi.org/10.1016/j.infsof.2017.11.015

[9] Malin Jansson, Stefan Hrastinski, Stefan Stenbom, and Fredrik Enoksson. 2021. Online question and answer sessions: How students support their own and other students' processes of inquiry in a text-based learning environment. *The Internet and Higher Education* 51 (Oct. 2021), 100817. https://doi.org/10.1016/j.iheduc.2021.100817

[10] Mohamad Kassab, Joanna DeFranco, and Valdemar Graciano Neto. 2018. An Empirical Investigation on the Satisfaction Levels with the Requirements Engineering Practices: Agile vs. Waterfall. In *2018 IEEE International Professional Communication Conference (ProComm)*. IEEE, Toronto, ON, Canada, 118–124. https://doi.org/10.1109/ProComm.2018.00033 ISSN: 2158-1002.

[11] Henrik Kniberg and Anders Ivarsson. 2012. Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

[12] Jan Knobloch, Jonas Kaltenbach, and Bernd Bruegge. 2018. Increasing Student Engagement in Higher Education Using a Context-Aware Q&A Teaching Framework. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, Gothenburg, Sweden, 136–145.

[13] Mike Neary, Gary Saunders, and Dan Derricott. 2014. *Student as Producer: Research-Engaged Teaching, an Institutional Strategy.* Project Report. The Higher Education Academy, York.

[14] Maria Paasivaara, Dragoş Vodă, Ville T. Heikkilä, Jari Vanhanen, and Casper Lassenius. 2018. How does participating in a capstone project with industrial customers affect student attitudes?. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training.* ACM, Gothenburg Sweden, 49–57. https://doi.org/10.1145/3183377.3183398

[15] Roger S. Pressman and Bruce R. Maxim. 2020. *Software Engineering: A Practitioner's Approach.* (ninth edition / roger s. pressman, ph.d., bruce r. maxim, ph.d., international student edition. ed.). McGraw-Hill, New York, NY.

[16] Beatriz Pérez and Ángel L. Rubio. 2020. A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 309–315. https://doi.org/10.1145/3328778.3366891

[17] Abdallah Salameh and Julian M. Bass. 2019. Spotify Tailoring for Promoting Effectiveness in Cross-Functional Autonomous Squads. In *Agile Processes in Software Engineering and Extreme Programming – Workshops*, Rashina Hoda (Ed.). Springer International Publishing, Cham, 20–28.

[18] Christopher Scaffidi. 2018. Employers' needs for computer science, information technology and software engineering skills among new graduates. *International Journal of Computer Science, Engineering and Information Technology* 8, 1 (2018), 1–12.

[19] Valeria Seidita, Massimo Cossentino, and Antonio Chella. 2012. A Proposal of Process Fragment Definition and Documentation. In *Multi-Agent Systems (Lecture Notes in Computer Science)*, Massimo Cossentino, Michael Kaisers, Karl Tuyls, and Gerhard Weiss (Eds.). Springer, Berlin, Heidelberg, 221–237. https://doi.org/10.1007/978-3-642-34799-3_15

[20] David H Smith IV, Qiang Hao, Vanessa Dennen, Michail Tsikerdekis, Bradly Barnes, Lilu Martin, and Nathan Tresham. 2020. Towards Understanding Online Question & Answer Interactions and their effects on student performance in large-scale STEM classes. *International Journal of Educational Technology in Higher Education* 17, 1 (Dec. 2020), 20. https://doi.org/10.1186/s41239-020-00200-7

[21] Rafal Włodarski, Aneta Poniszewska-Marańda, and Jean-Remy Falleri. 2022. Impact of software development processes on the outcomes of student computing projects: A tale of two universities. *Information and Software Technology* 144 (April 2022), 106787. https://doi.org/10.1016/j.infsof.2021.106787