-Objectives of the Software Report

\* Practical goal of the project (e.g., standard, new, unusual)

e.g. identification or analysis of existing problems, finding solutions, or implementing them in practice, or evaluating the results of the implemented solutions.

-Methods/theories that were used to achieve objectives/practical goals

\* Novelty of the theory/method (e.g., standard, new, has not been applied before)

When some specific methods/theories were used, the report should explain why they were chosen. For example, "the method is widely accepted in the software project management community and the team members were quite comfortable with it".

E.g we had to use architecture diagram BUT explain why we chose MVC

\* Business/social context of the work(e.g., investigated many times before, less known)

-Outcomes and lessons learned

\* Outcomes/Lesson learned/Conclusions (e.g., well-known, new, contradictory)

Positive and negative

-Conclusions derived from the experience

-Don't introduce anything new

-Conclusions should logically follow from the outcomes of the project and lessons learned. These conclusions should be presented in some generalized form so that they can be compared with the results from other experience reports or tested by researchers doing "case studies".

NOTE

-The report should not be written as a "diary". The results should be presented in some generalized form, omitting small details that can be considered as very specific for the current context. The results should be explained on the level of abstraction that is suitable for your contexts.

-The experience report is to be "readable" for practitioners, not only for researchers.

-References in each statement should be avoided. The part of the paper that describes methods/theories chosen should have relevant references. If the researcher chooses, there can be a separate "related works" section at the end of the paper.

**Intro**

In the modern world any technology based undergraduate study “requires not only studying theory using textbooks, but also providing students with the experience of typical non-technical issues in a software project.” [1] Behind software implementation lies an enormous domain of theoretical frameworks, methods and concepts. While we have readily accepted these in teaching, following our practical experience we have come to question how well these theories really reflect what is applied in practice. The objective of this software report is to determine whether the theory of software development processes and practices accurately describes how these methods are carried out in practice. In doing this we will outline the theory of specific software development methods and describe our experience carrying out these methods in the context of our undergraduate study group software development project. We will then compare the two in order to determine whether software methods apply in practice in the sense they are described in theory. In essence we will be demonstrating the differences between the theory and practical implementation and what this means for our future projects.

[1] Gnatz, M., Kof, L., Prilmeier, F., & Seifert, T. (2003). A Practical Approach of Teaching Software Engineering.<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.8390&rep=rep1&type=pdf>

**Ryan**

**Requirements Engineering:**

When given a new project, teams need to start somewhere, software development theory states knowing what requirements are needed is the first step [2]. This is exactly what we did to start off our project. Planning and understanding the project given is key to success of any new venture. User stories are a description of a feature told from a user and are commonly used and extremely good at describing the system and how everything falls into place. We therefore decided to use them in the context of our project, as the client had little technological understanding and additionally we have had success in using them in the past. While we wrote up many user stories to start with, we soon found we still had not covered all of our system and continued to write more to fully grasp the whole scope of the system. Our experience therefore conforms to requirement engineering theory as we learnt the importance and value in thoroughly carrying it out. Identifying stakeholders to use in user stories is also very helpful. Conventionally, figuring out your stakeholders is important as to understand your system better, and figure out what is required [3].

<https://dl.acm.org/doi/pdf/10.1145/1044834.1044837?casa_token=YVsgSoKdGF4AAAAA%3A75BdcB8ELewzGF8r8ZpKUFGDghe5zg2tGCU0_W15VnEZanQRzUrKfoC02BwQP-mIpeS39BXFxVyX>

[2] Ian Sommerville and Jane Ransom. (2005). *An empirical study of industrial requirements engineering process assessment and improvement.* ACM Trans. Softw. Eng. Methodol. 14, 1 (January 2005), 85–117. DOI:https://doi.org/10.1145/1044834.1044837

<https://discovery.ucl.ac.uk/id/eprint/744/1/1.7_stake.pdf>

[3] H. Sharp, A. Finkelstein and G. Galal. (1999). *Stakeholder identification in the requirements engineering process*. Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99, 1999, pp. 387-391, doi: 10.1109/DEXA.1999.795198.

**Project Estimation:**

After completing requirements, theory states you need to carry out project estimation. Project estimation helps plan how the system is going to be built, by weighting the time and resources of each use case [4]. In our project, we found that using the method of planning poker worked very well. Planning poker lets all members put forward an estimate for each use case, without discussion at first, to give the team an idea of how each member views the difficulty of the task without others opinions [5]. We chose this method because it is widely accepted, and our team was very comfortable using it. Additionally it fit well in the context of our work being an undergraduate group project as it was not too complicated and incorporated all group members' contributions. Discussion is then given, and a final estimate is achieved. In our experience, we found that we generally agreed with the theory and saw the benefit in carrying it out this way. We learnt that our ability to estimate would improve with practice. For example we had a few discussions on some of the weights, such as for sentiment analysis as we were unfamiliar with it. Some of us thought that it would take 20 story points, others thought it would take even longer, at 40 story points. The theory and how it is carried out in practice was therefore similar in this case.

<https://www.researchgate.net/profile/Roy-Clemmons/publication/200036324_Project_Estimation_With_Use_Case_Points/links/00b7d518401cd4fd9b000000/Project-Estimation-With-Use-Case-Points.pdf>

[4] Roy K. Clemmons. (February 2006). *Project Estimation With Use Case Points*. <https://www.researchgate.net/profile/Roy-Clemmons/publication/200036324_Project_Estimation_With_Use_Case_Points/links/00b7d518401cd4fd9b000000/Project-Estimation-With-Use-Case-Points.pdf>

<http://athena.ecs.csus.edu/~buckley/CSc131_files/Planning%20Poker.pdf>

[5] Mike Cohn. (2007). *Planning Poker*. ​​http://athena.ecs.csus.edu/~buckley/CSc131\_files/Planning%20Poker.pdf

**Caleb**

**Project Planning and Scheduling:**

In order to identify how we would produce our project we employed a feasibility study. A feasibility study is used to brainstorm on if and how the majority of requirements would be completed [6]. In our case we discussed how a requirement would be achieved (if we knew), who was proficient in what, and who was interested in producing parts that we had no experience in. Beyond that, the feasibility study allowed us to bring up follow up questions to the client, such as regarding how he would like the project to be deployed and the legal status of this work. We followed the traditional model for creating a feasibility study, because as a team we were familiar with the method (from previous information papers) and it is well used in theory. This was helpful in constructing an outline of what needed to be addressed and constructed. Allowing us to prepare the necessary tools to complete said tasks, reducing the time needed to complete the project. By ensuring we were comfortable with what was chosen we could ensure we left unwanted work down to a minimum for the team.

Taiga is a tool that is used to allocate User stories and SPRINTS to certain time periods and developers [7]. While we have used other tools to do this in the past (Excel), it was new to the whole team to use Taiga as a tool for SCRUM management.

As a team we did not follow the user stories implementation exactly the way that would be employed in theory by experts. This meant we had to re-do our user stories from time to time, as they were not properly descriptive of the circumstance. We were unaware of the exact depth that we would have to think from a user's standpoint. A lot of user stories were added to the sprints as time passed, as practically developing the project allowed us to see parts we were blind to beforehand, due to the lack of experience. We also have learnt that we are more set on impressing the client rather than the users on average. While we have added user stories to help the average user, the importance of hitting the clients requirements can be in fact conflicting.

Lastly we decided to ensure 2 breaks within our sprints. Our collective previous experience tells us we may fall behind on certain workloads. Thus this break will be used to tie up loose ends of producing our project. This is therefore an example of when we deviated from theory and it worked well.

Agile is an approach to project management and software development. Within agile, the requirements of the project are developed in increments that are often pre-planned. Using frameworks such as SCRUM to help promote teamwork and techniques such as SPRINT to help divy up work into fortnightly segments.

We can identify 4 main reasons why agile was the method of choice for our project.

1: Previous Experience of Agile within the team, 2: Time management, 3: Uncertainty of project requirements, 4: Methodology being widely used in the industry [8]. By using the wisdom of the industry , we can conclude that given such reasons agile is a useful methodology for projects that can change rapidly. Due to the context of the project, in which requirements could have been added or dropped at anypoint of the development cycle, Agile is known to be the perfect methodology for such scenarios.

The choice of employing Agile has been quite useful as of 17/09/2021. The framework/theory allowed us to stay structurally sound and on task. Whilelockdown has slowed down our progress somewhat, however, planning our project using SCRUM and the flexibility of SPRINTS has given us ample room to adapt to this new circumstance.

By using method systems such as SCRUM and tools we have known to work, we were able to cut down on the production plan [9]. Collectively having expertise on the same method we were able to move forward on a unified and efficient basis. We have therefore learnt the importance of planning and scheduling as outlined by the theory.

Within agile and SCRUM there is a process called SPRINTS [10]. A SPRINT allows us to divide up the work envisioned in the process of developing the app into fortnightly segments. But due to this short timeframe, it allows the team to be flexible and change what needs to be completed.

**[6]**Orsmond, G. I., & Cohn, E. S. (2015). The Distinctive Features of a Feasibility Study. *OTJR: Occupation, Participation and Health*, *35*(3), 169–177. <https://doi.org/10.1177/1539449215578649>

[7] M. Cohn, User Stories Applied, Addison-Wesley Professional, 1/03/2004.

[8] Consultancy.eu. (2020, May 7). *Half of companies applying Agile methodologies & practices*. https://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodologies-practices

[9]**Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.**

**[10]Sommerville, I. (2011). *Software Engineering 9* (9th Ed.). Pearson Education.**

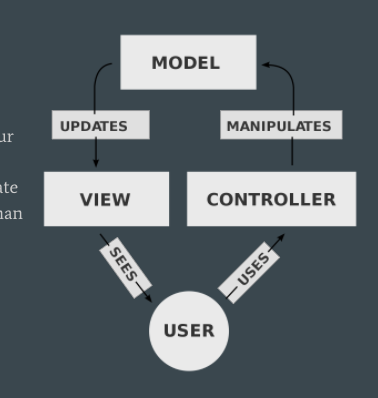
**BILLY**

**System Modelling:**

According to [11], System modelling is a crucial part of the software development lifecycle as it helps the many developers within a team to gain a shared understanding of the system, and share this understanding when communicating with customers. This then provides clarity and confirmation of the system specification, ensuring a successful implementation. System modelling can be effectively used on existing and new systems, each time serving a slightly different purpose [12]. There are a variety of different perspectives from which a system can be modelled, depending on the situation that requires it to be modelled. In our project, we have made use of mainly behavioural and structural models, we will discuss why below. Respectively, behavioural models are used to model the dynamic behaviour of a system during execution, while Structural models describe the organisation and relationships among system components [13]. For behavioural models we made use of a sequence diagram, and for structural we used an ERD and class diagram. This was because they were the only ones relevant in the context of our work. Interaction models were not necessary as we used other methods to identify user requirements. External models were not used as our project will not be interacting with any other systems, as it is a standalone solution. We followed common system modelling practise for the creation of the ERD and sequence diagram, which resulted in diagrams that allowed us to guide our implementation and did not require reiteration. However, proper practise was not followed for the creation of the class diagram, which resulted in issues. This meant we had to redo the class diagram and the domain class and DAO work we had already done to accommodate these changes, which delayed our implementation. This conforms to theory around system modelling, and demonstrates how we learnt importance of consistent adherence to the model theory, otherwise, as in our project, issues can arise.

**Architectural Design:**

As stated in [1], architectural design is used to aid in the design of the overall structure of the system. It allows the project team members to understand how the system should be organised, and serves as the key link between the design and requirements engineering processes [14]. This is because of its ability to identify the key components of the system, and the relationships between them. Theory outlines that the key benefits of architectural modelling are the ability to conduct focused stakeholder communication, perform system feasibility analysis for the non-functional requirements and the ability to reuse architecture for a range of systems [15]. Within the architectural design sphere, there are a range of patterns that conform to common software practise and have strong evidence supporting their use [16]. One such pattern is the MVC or model-view-controller pattern, which our team made use of. We chose to use the MVC pattern as our system fit in nicely with the scenarios it is best utilised in. For example, MVC is used to speed up development time, and makes it easy to have multiple developers working on the system at once. In the context of our project, there is a strict deadline of ten weeks, and our scrum style of development means we are frequently working on the system at the same time. In our project, we firstly used the MVC pattern to create models for each model,view, and controller. Respectively, these were an ERD diagram, lo-fi prototypes of the webpages, and the Class diagram. We have also utilised the MVC pattern to maintain good separation between our front-end, back-end, and html sections. The experience with MVC so far has been a positive one, making use of the pattern has not yet caused any detriment to our project, in fact the opposite. This is a testament to the theory of architectural design, and the use of software patterns in general. From our experience with MVC and architectural patterns in general throughout this project, our team has learnt that they aid greatly in the development of web based applications and will be making use of them in the future.



**Software Implementation:**

Software implementation is the part of the software development lifecycle where executable software is developed [16]. In an agile process, such as our project, this phase is interconnected with the design phase in an iterative process that blends design and development to create systems that truly satisfy stakeholder needs. This is where the models created in the system modelling and architecture design phases are translated into actual software [17]. Throughout the implementation phase there are a variety of considerations that must be taken into account. Firstly it is important to consider what parts of the software can be reused from other systems we have already created, and what parts could be reused in the future [18]. In our project, we reused much of the backend, with a variety of the DAOs largely coming from our previous 310 projects, and being altered to fit our needs here. Secondly configuration management must be considered, especially in projects such as ours where development is distributed. We have made extensive use of git-bucket as this was familiar with all team members and we felt switching provided no obvious benefits. Finally there is the consideration of Host-target development, which [19] states requires consideration as software is often developed in a programmer environment, which differs significantly from a client environment. However, in the context of our project, we have found this relatively unnecessary to consider, as most of the development is being done in a very similar environment to the client, with our team working on relatively low powered laptops, so testing of the system happens in an environment you could reasonably expect the client to be in. We have therefore learnt how and why taking into consideration these factors is an important and common consideration in the implementation process. Doing so has allowed us to avoid delays to our project and develop at pace through reuse. Our success in implementation reinforces the theories of software implementation and shows how adherence has positive effects on the software development lifecycle.

[11]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[12]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[13]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[14]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[15]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[16]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[17]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[18]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

[19]Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

**Latif**

**Risk Management and People Management:**

Risk management is a crucial concept in the success of a software development project. If a significant risk is not captured and addressed in the early stages of development, it could lead to the project's failure. Reference [1] states most of the problems in software projects could have been averted if high-risk elements are identified and resolved early on [20]. Reference [21] sets out a generic agile approach to risk management to ensure a project can continue in spite of risks. This included planning risks and mitigation responses. We followed this approach as this is commonly used and it is what we had success with in the past. It also works well in the context of a short 10 week project as we do not want to waste lots of time which could be avoided through risk management. The team has documented a few high risk issues that we might face and promptly acted on it by tasking a member to do research and testing, while keeping an eye for other risks that might occur as the project continues. Our experience conforms to the theory as in doing this risk management we have managed to avoid having any major issues that we don't know how to address. We have therefore learnt the importance of risk management.

[20] Kwaka, Y. H., & Stoddard, J. (2003). Project risk management: lessons learned from software development environment. *Technovation,* 24(11), 915–920.

[21]Moran, A. (2014). *Agile Risk Management* (1st ed.). Springer International Publishing. https://doi.org/10.1007/978-3-319-05008-9

People Management:

Theory outlines that people management plays a major role in the success of both small or large software development projects. Success of a project depends on the skills, motivation and interaction of people. Without good management, the outcome of the project is inadequate [22]. In an agile approach the team has to be motivated, honest, inclusive of all members views, consistent and respect each members strengths and weaknesses.

Our approach to people management was consistent with this agile approach as described in theory. Throughout the project, each member chose tasks they want to work on based on their strengths. We had discussions everyday through facebook messenger and progress meetings once or twice a week to assign new tasks and get help or feedback on issues. In conforming to the agile principles, we learnt how helpful it is to get everyone on the same page while tapping on the strengths and creativity of the team. We therefore learnt how much more smoothly a project can run if you have good people management, this is therefore in line with people management theory.

<https://www.researchgate.net/publication/324052196_People_Management_in_Agile_Development>

[22] Thiago Rocha De Alcantara, P., Dias Canedo, E., & Parente Da Costa, R. (2018). *People Management in Agile Development.* https://www.researchgate.net/publication/324052196\_People\_Management\_in\_Agile\_Development

**Software Quality (including Usability), Measurements:**

Software quality is a broad subject comprising understandability, usability, security, efficiency, applying standards, and testing. [23] All software theory outlines the importance of these components in creating high quality code. Standard and common software quality methods include creating development, release and user testing. Testing is essential to confirm the software’s quality, whether certain functions work as intended and no bugs are present. We do not have a dedicated class file for testing purposes at the moment as we decided to write all code before writing test classes, however we have used peer review and have planned to complete testing before the end of our project.

Theory outlines usability and understandability are other important aspects of software quality. Usability ensures a user interface is easy to navigate, needs little to no effort to learn to use, and has minimal cluttering with a professional look. Understandability comprises four main components: complete, concise, clear, and organised software. Currently our project is not yet complete, nor concise, but it is clear with good and consistent naming conventions and organised source codes. We therefore believe it will become understandable and usable in the weeks to come. Security is another important aspect of software quality. We do not want unauthorised users to be able to execute actions or access sensitive information. During the first weeks of the project, we found a security risk in the lo-fi prototype which would lead any new user to register with the highest authority and due to this, we therefore reviewed the other prototypes to avoid any other security risks. Therefore, while our project to date does not conform to all theoretical practices surrounding software quality, we believe the code we have produced so far is of high quality and by applying further testing and quality methods going forward we will be able to get it to that stage by the end of our project.

[23] Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.

**Sarah**

**Software Configuration Management (SCM):**

Effective SCM is an essential method for software projects of all sizes. It ensures effective and smooth communication and development, and allows developers to work in parallel [24]. Small projects with lots of developer communication, such as in the context of our project do not require complicated SCM processes. This would increase costs and slow things down unnecessarily. [25] Theory however outlines SCM is very important for small teams as they often lack defined processes [26].

Throughout this project, we carried out standard and well used configuration management activities suitable for a small project. This included version and change management and system building. We used Git-Bucket for our central repository, as we were proficient using it and our mentors could track what we were doing. In terms of change management we learned the importance of placing a strong focus on documentation, doing so for all requirements at the beginning and then recording changes as we went. At the beginning of the project we did not stick to version management practices strictly, this led to an early merge conflict. Following this we learnt to place a larger focus on adhering to these practices. This included conforming to agile principles and making small, frequent changes and commits and using branches. Additionally checking others code before merging and working on separate components and communicating when not to work on certain files. Since doing this, we have not had any merge conflicts. Our experiences therefore conform to the conception in theory that software configuration is an essential method for software development projects. They also support that small projects do not need large configuration management tools in order to seek these benefits.

**Professionalism and ethics**

Reference [27] recognises that professionalism is crucial in the IT profession. Information system professionals need to understand ethical imperatives in order to carry out their work [28]. If a professional sign up to a Code of Ethics they must strictly comply with the ethical guidelines. In the context of this project we are not working professionals. However, [29] outlines that for graduates to become professionals they need professional attitudes, be able to follow good practices and know how to work in a team. Group projects at an undergraduate level assist in developing this. Theory therefore states it is important that students adhere to these when carrying out software projects in order to prepare for the real world. We therefore decided to take a standard approach to ethics and professionalism for undergraduate group work in taking these factors into account, but not following the rigid guidelines. Professionalism included working as a united team, actively contributing, being transparent, good and frequent communication, staying on task during meetings and being prepared, focused and confident when presenting progress. Maintaining good time management is also important, for example, arriving on time, getting work done and not over committing and constantly updating documentation.

Reference [30] highlighted that one issue with undergraduate group projects is that codes of practice are not addressed. Our experiences conform to this as we did not have to strictly adhere to ethical guidelines. We however did make ethical considerations in our project. This included confidentiality and respecting client privacy even though we haven't signed any agreements, not saying we can do things beyond our competence, and ensuring to correctly implement features requested by the client to comply with IP law. We therefore learnt the importance of professionalism and ethical principles in our project in preparing us for future work. From our experience the theory therefore aligns with this method in practice.

**Process Improvement and Retrospections**

Reference [31] outlines that end of iteration retrospections are an important part of agile software development to reflect and seek future areas of improvement. Retrospection enhance the users’ engagement and increases their awareness about work, collaboration with others and productivity [32]. We took an informal approach to retrospection as this is a small short-term group project for University. While the industry is motivated to ensure software does not fail for customers, in the context of our work we are wanting to get the best grades possible and are not motivated to improve the processes for long term development. Therefore rather than formal retrospection, we did peer inspections to check each other's code and reflected sporadically. This ensured defect removal, progress assessment and quality related evaluations. However, it meant retrospection was limited to code not processes. We aimed to take an agile approach to process improvement as we were carrying out iterative development. This is a standard and well researched method, with many benefits in increasing productivity and product quality. However due to lack of motivation to improve processes for a short term project as well as communication issues with our client, we have been unable to regularly make improvements to our processes. We have however made some improvements from sporadic evaluation. For example, allowing anyone who is available to check the code before merging rather than having one owner who checks all of it.

This approach was not that common or investigated, we therefore learnt that we would have benefited from more frequent and formal iteration retrospections. These are as much of an important part of software development in practice than they are set out to be in theory. In order to seek these benefits software development teams must commit to making regular retrospections and improving their processes even though it doesn't have a direct impact on the completion of the project. While it may appear tedious and time consuming, it will evidently be more efficient and produce higher quality code.

[24] Berczuk, S. P., Berczuk, S., & Appleton, B. (2003). *Software configuration management patterns: effective teamwork, practical integration* (3rd ed.). Pearson Education.

[25] Berczuk, S. P., Berczuk, S., & Appleton, B. (2003). *Software configuration management patterns: effective teamwork, practical integration* (3rd ed.). Pearson Education.

[26] Berczuk, S. P., Berczuk, S., & Appleton, B. (2003). *Software configuration management patterns: effective teamwork, practical integration* (3rd ed.). Pearson Education.

[27] Humphrey, W. S. (1996). *Managing technical people: innovation, teamwork, and the software process* (1st ed.). Addison-Wesley Professional.

[28] Mingers, J., & Walsham, G. (2010). Towards ethical information systems: The contribution of disclosure ethics. *MIS Quarterly,* 34(4), 833–854.

[29] Myers, C., Hall, T., & Pitt, D. (1997). Student Projects and Professionalism. *The Responsible Software Engineer.* Published.

[30] Humphrey, W. S. (1996). *Managing technical people: innovation, teamwork, and the software process* (1st ed.). Addison-Wesley Professional.

[31] Drury, M., Conboy, K., & Power, K. (2012). Obstacles to decision making in agile software development teams. *Journal of Systems and Software*, 85(6), 1239–1254.

[32] Meyer, A., Murphy, G. C., Zimmermann, T., & Fritz, T. (2017). Design recommendations for self-monitoring in the workplace: Studies in Software Development. Proceedings of the ACM on Human-Computer Interaction, 1, 1–24.

**Conclusion/future prospects:**

In conclusion, based on our experience software configuration management theory is in line with how these methods are applied in practice. We used a blend of closely following theory and deviating from the norm. We did have positive results in acting outside the norm in areas we had proficient experience beforehand. However our biggest finding was that in the areas in which we did not strictly adhere to the methods were the areas where we ran into issues. This therefore demonstrates that the theory is correct in stating the importance of these methods. However more effort needs to be put into ensuring these methods were correctly applied to gain the benefits of the methods as described in theory. This is of vital importance as we learnt how these benefits were easily lost when deviating from the process described in theory. To further this research we would suggest taking a large sample of practical implementation experiences into account to observe how strictly they adhere to the methods. Perhaps the findings would be different if the practical implementation was carried out as part of a paid job rather than a university project.