Designing for Real People: Teaching Agility through User-Centric Service Design

Robert Chatley
Dept of Computing
Imperial College London
London, United Kingdom
rbc@imperial.ac.uk

Tony Field
Dept of Computing
Imperial College London
London, United Kingdom
ajf@imperial.ac.uk

Mark Wheelhouse
Dept of Computing
Imperial College London
London, United Kingdom
mjw03@imperial.ac.uk

Carolyn Runcie
School of Design
Royal College of Art
London, United Kingdom
carolyn.runcie@rca.ac.uk

Clive Grinyer
School of Design
Royal College of Art
London, United Kingdom
clive.grinyer@rca.ac.uk

Nick de Leon
School of Design
Royal College of Art
London, United Kingdom
nick.leon@rca.ac.uk

***Abstract*—We present the design and evolution of a project-based course – Designing for Real People – that aims to teach agile software development through an unwavering focus on the user, rather than emphasising the processes and tools often associated with a method like Scrum. This module is the result of a fruitful collaboration between a Computer Science Department, bringing knowledge and skills in the software engineering aspects, and the Service Design group of a neighbouring Art College, with expertise in user research and user experience design.**

We present the details of the current structure, content and assessment strategies developed for the module, as well as the principles behind its design. The core theme of the course is gathering and responding to feedback, and so here we present how this has been applied to the design of the module itself, with lessons learned, and improvements made over time. By reflecting on our own work, we aim to provide recommendations that may aid others considering how to teach these topics.

***Index Terms*—education, agile development, service design**

I. Introduction

The first principle of the Agile Manifesto [1] states that "our highest priority is to satisfy the customer through early and continuous delivery of valuable software". As with many universities, over recent years we have developed and evolved modules that aim to teach the mechanics of agile software development, with student teams working to deliver a system to meet a customer's requirements. On completion of these projects, the student teams typically delivered sophisticated pieces of software, often integrating multiple technologies, which was impressive. However, we realised that the students' focus was often on the technical complexity of the software produced, regardless of how well it met the users' needs.

At the same time, a common theme in student feedback from end-of-course surveys (from around 200 students annually, over a period of three years) was that agile practices felt like baggage and distraction to them. Our teaching had begun to centre on the mechanics of agile methods and the software engineering process rather than the underlying principles. Students reported having to commit a lot of time to meetings and points of process (and documenting these for assessment) which they felt was taking time away from "actually getting on with the project" by which they meant "writing the software". By requiring students to complete the rituals and ceremonies associated with a process like Scrum, and assessing them on how they were doing these, we had diverted from the original spirit of agile development. While we still believe that agile methods are vital for modern software projects, we feel our mistake was to focus on students completing the process "correctly" rather than on satisfying the customer or delivering early and continuously.

This feeling that a focus on how agile practices are carried out can be an obstacle to completing a project effectively is not unique to the classroom. The use of "agile" in industrial software development continues to grow [2], but the practices and processes adopted by many teams are far away from what the authors of the Agile Manifesto originally envisaged. Allen Holub summarises this well when he writes: "The word *Agile* has taken on a meaning that the folks who coined the term at Snowbird wouldn't recognize—a rigid process, mindlessly followed by a small part of an organization that isn't the least bit agile (lower case). The word agile means nimble, flexible, adaptable, quick. If you aren't all of those things, you're not agile, whether or not you're using some "Agile" technique. Agile is a frame of mind and a culture, not a process. You can implement Scrum perfectly, and not be in the least bit agile. What's important, then, is *agility*, not Agile™." [1]

It may be natural for computer science and software engineering students to be excited about developing new pieces of software and solving difficult technical problems. Similarly it may be easy for educators to fall into the trap of teaching and enforcing a rigid process under the name of agile development, but we felt the need to take a step back. We want to focus on

[1]See Holub's "Getting Started with Agility" at https://holub.com/reading/

the principles of agility. While we do want to encourage the use of an iterative design process, and we do want the students to develop software, we want to shift their mindset away from the process and the technology, to focus on finding the right problem to solve, following user feedback. Coming back to the first principle of the Agile Manifesto, we want students to use the continuous delivery of *valuable* software to *satisfy customer needs*. Emphasising user needs led us to think less about the technicalities of software development, and more about *Service Design* [3]. Service Design is a human-centred design method used to design and develop new services across technological, social and environmental projects in commercial, social and public contexts. It is not explicitly a software engineering method, but with the digital transformation of so many aspects of daily life – from paying your taxes to buying a bus ticket – designing services frequently involves a digital element. This often requires the development of new software, and so software engineering and service design become intrinsically linked.

In this paper we describe a project-based module called *Designing for Real People* (DRP), which aims to teach the underlying principles of agility and agile software development through an unwavering focus on the user. The module has been developed and is run as a collaboration between the Department of Computing at Imperial College London, and the School of Design at the Royal College of Art (RCA). Taking a *Service Design* angle on a university software engineering course led us to build a project-based module where teams create digital products led by human-centred design, user research techniques, prototyping, and iterative feedback. We underpin these activities with effective software engineering techniques, but these are always in a supporting role. Responding and adapting to user feedback is always the primary goal. We want the students teams to be agile, not to "do Agile".

As an experience report paper the main contributions are a description of the module, including details of its rationale, evolution and implementation, and the insights gained after a number of iterations on to the way the module is taught and assessed. These insights can be summarised as follows:

- We believe it is more important and effective to focus on teaching the underlying principles of agility rather than any specific agile method. At the scale of a university project, requiring teams to perform some of the ceremonies associated with common agile methods can feel like overhead to students. This can give the impression that agile methods get in the way of delivering a software project, rather than helping.
- An assessment framework based around process and ceremony may be easy for educators to implement, but it can send the wrong message, and can lead to students completing certain rituals just to tick a box, rather than helping and encouraging them to deliver better software. We recommend structuring assessment around whether improvements to the software are delivered iteratively and, crucially, how the product has evolved in response to feedback gathered from real users.
- We have found that team projects aimed at teaching agile methods and thinking work best when guided and assessed by both software engineering experts and individuals whose primary expertise is in a non-software related field. Here, we focus on a collaboration with experts in Service Design [4], but we believe that the same ideas can be applied to other user-focused disciplines such as manufacturing, healthcare or education. Paired tutoring, where experts from both disciplines work together as equal partners, ensures that students pay equal respect to both the enabling technology and user needs.
- Aiming to teach tools and techniques for software engineering at the same time as those for effective user research and design in a single software engineering class can be overwhelming for students. Our recommendation is to separate the teaching of practical technical aspects – for example those related to automated testing and continuous integration – into separate units that are covered before a team project starts. This allows students to focus on their core mission in the project of iteratively delivering a product to their intended users, without having to master new technical concepts at the same time.

The following sections discuss the principles of agility as outlined in the Agile Manifesto (Section II) and background in Service Design (Section III), which have influenced the design of the DRP module. Details of the module's evolution, implementation and assessment are given in Sections IV, V and VI. We conclude with a discussion of our reflections, lessons learned and advice for other educators.

II. Reflecting on The Agile Manifesto

The Agile Manifesto [1] is a document written by a group of eminent software engineering practitioners to characterise their views on good practice in software development projects. This has been the basis for a lot of subsequent work on different aspects of "agile software development". The manifesto sets out four values and twelve principles. The values take the form of "We value X over Y". This should be taken as meaning that although the authors do find value in Y, and it should certainly not be discounted, they find greater value in X.

In developing the DRP course, we did not set out explicitly to address each of the values set out in the Agile Manifesto. However, it is interesting to reflect on the fact that by taking a focus on the user as our north star, what has emerged is very much in line with these four values:

A. Individuals and interactions over processes and tools

It is easy, in trying to teach agile methods, to focus on processes and tools. These are things that are relatively concrete. For example, it is easy to describe the ceremonies associated with the Scrum process, and to check that student teams are carrying them out. It is easy to ask students to demonstrate the use of particular tooling to manage a backlog of tasks, and to require them to cross reference code changes with tickets and issues. There are many tools that support automated testing, and we can use metrics like test coverage to verify that these have been employed by teams. But, none of these are inherently agile, and in fact the Agile Manifesto values "individuals and interactions" over processes and tools.

Interactions between individuals are much less constrained than defined processes. This is what makes them valuable. Developers, users, sponsors and other stakeholders all interacting freely in a high-bandwidth manner allows us to steer the product developed to meet the user's needs as closely as possible, and to deliver maximum value and impact. However, in a classroom setting, effective interactions among individuals are much more difficult to promote and assess. One of the key challenges has therefore been to create a learning experience where students are encouraged to focus on individuals and interactions, both through the message being delivered by the course leaders, and through the assessment structures that frame their work and incentivise their behaviour. Although processes and tools can certainly be useful in organising work and controlling software quality, the primary goal is to deliver something that is of value to the user.

B. *Working software over comprehensive documentation*

We have evolved our software engineering teaching over a number of years and across the curriculum. As part of this, we have reduced the amount of documentation that students are required to produce. For example, we used to ask teams to write documents setting out lists of requirements, their testing strategy, or describing how they were applying a particular agile method. However, feedback from students suggested that the time spent writing these documents was seen as a distraction and took away time that could have more usefully have been spent "working on the project".

In the latest iterations of DRP, we have kept documentation deliverables to a minimum, and now focus assessment and tutor feedback on the demonstration of working software. Evidence that the students have engaged usefully with their target users comes in the form of an additional, but lightweight, portfolio of notes, audio recordings, video clips etc.

C. *Customer collaboration over contract negotiation*

Reducing the emphasis on documentation has meant moving away from writing a requirements specification at the beginning of a project, or asking students to estimate and commit to what they will deliver by the end. Instead we have moved to a model that stresses the importance of developing their solutions iteratively and incrementally, and to involve the user at every stage, adapting the product based on their feedback.

We have specifically avoided an assessment scheme where the deal is "if you build X, then you will get mark Y", or "you said you were going to build X, but you didn't build X, so you didn't meet your goal". Instead we want the students to get as close as possible to solving their users' problem, and if this means changing direction and pivoting their idea one or more times during development, that's fine.

D. *Responding to change over following a plan*

Following on from the above, we also do not ask teams to set out a plan for development at the beginning of their project. Even if the students successfully break down their initial objective into milestones for incremental delivery, and complete those milestones as planned, this is no good if they are not engaging with their users and responding to feedback. We would much rather see a project that changed direction in response to feedback and ended up delivering something completely different, but more valuable, as a result of this, over one that planned meticulously and delivered on time and on budget, but without refinement through continuous feedback.

That said, we should not completely discount following a plan, at a smaller scale. We have found that teams that are able to organise themselves, and who plan effectively on a shorter timescale, are able to pull together to deliver more meaningful increments more often, which has in turn yielded more opportunities for demonstration and feedback, and better final products. But, once again, talking to the user and responding to feedback is the key – everything else supports that.

III. Service Design

One of the main techniques that we use in the DRP course to promote design thinking is the Double Diamond. The Double Diamond [5] is a framework originally created by the UK Design Council that visualises design as a problem solving tool. The initial aim was to allow this application of design to be easily understood and applied by businesses, public and third sector organisations. The separation into two diamonds in the visual representation aims to highlight the importance of initially spending time and resources on understanding the problem that an eventual design may later try to solve.

The Double Diamond comprises four different stages of a project: Discover, Define, Develop and Deliver (see Figure 1), and is supported by a variety of practical methods. Different methods may be used at different points in the process to help stakeholders gain increased understanding of a challenge, generate and then test new ideas to create new forms of value and improved outcomes.

The course introduces students to design and technical tools, skills and approaches that we hope will help teams to develop innovative experiences for people in their everyday lives. To unite design and software engineering, we focus on building meaningful digital interactions within a given context, and creating digital "touchpoints". Methods introduced during the DRP course include problem framing, stakeholders lists and maps, empathy and user journey maps, pen portraits, personas, prototypes, storyboards, affinity mapping and visioning tools among others.

Student teams work on a project in a problem area proposed by an industry partner, or in a domain that is a personal area of interest. Finding an appropriate context enables the team to focus on the challenges and opportunities faced by people in real scenarios. There is some nuance to finding a good problem area, and in particular we have found it important to vet external project proposals.

The chosen project should require teams to understand the people involved in that context, what they do and some of the

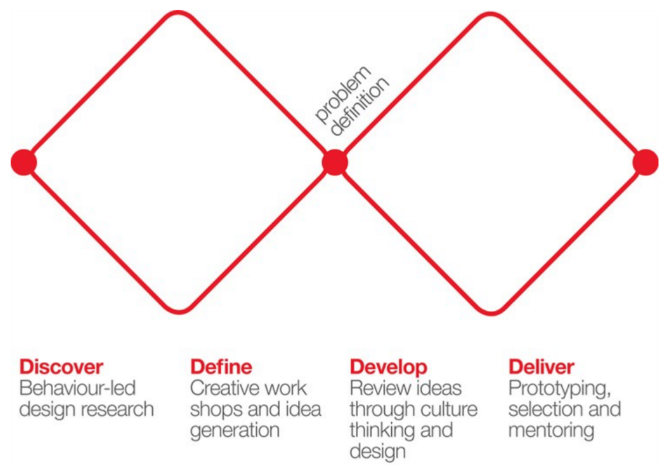
problem definition
Discover
Behaviour-led design research
Define
Creative work shops and idea generation
Develop
Review ideas through culture thinking and design
Deliver
Prototyping, selection and mentoring

Fig. 1. The Double Diamond design process - figure adapted from [5]

challenges they face in everyday life. Understanding this will inevitably reveal a variety of interactions (touchpoints), both physical (e.g. face-to-face in a range of physical environments) and digital (websites, apps, texts, phone calls) and using different digital devices. Consequently, students are expected to build a digital touchpoint within their chosen context, that will cater for the needs of their specific target audience. This typically takes the form of a web or mobile application.

IV. Interdisciplinary Collaboration

Combining the expertise of software engineers within the Computing department at Imperial with the design expertise of the Service Design group within the RCA has proved a very fruitful collaboration, but it has taken a while to get to the point where everything is aligned.

Initially we had two separate projects. One was a design project, run by staff from the Art College, concentrating purely on how users would interact with a system, and interface design. The other was a technical project run by staff from the Computer Science department, which involved building a web application backed by a database, and using this as the basis of an online game. In terms of the Double Diamond model, one project covered the first diamond, while the other covered the second, but there was no connection or progression from one diamond to the other. Students became frustrated at designing something that they did not go on to build. They also felt overloaded from working on two separate projects in parallel, and found it hard to balance their time between them.

With this realisation, combining the two projects so that the students first designed a system and then went on to build it seems an obvious next step. However, the feedback that we got on the combined project was not good. The students, being computer scientists, had a natural affinity with technical topics and with the staff from the Computer Science department. As a group, the students were not very engaged by the design project as they felt it to be "non-technical" and abstract. They came to university to study computer science and enjoyed working on technical problems. They respected the Computer Science faculty who spoke in a language that resonated with them, and shared some of their values. They felt less connection with the staff from the Art College, and this reflected in their views about the design project and the feedback that they got on their work.

When we combined the design and the technical projects, the students still had negative feelings about the design work. Technical and design aspects were assessed separately by the relevant staff based on their respective expertise. The students often felt that the design staff were giving feedback that was inconsistent with what the software engineering staff were saying, and they were upset when they had spent a lot of effort on a complex implementation, which the computing staff appreciated, but the design staff often did not (and hence they were often branded disparagingly as "non-technical" by the students). Students had a natural tendency to concentrate on areas that they did not like in their feedback. They were quite negative about the combined project, and often said that they would prefer a purely technical project.

Reflecting on this we noted that there was something of a *home-field advantage* for the software engineering staff. The students knew them, and felt more connection with them as part of their department, compared with the "away team" from another institution who were not computer scientists and often did not appear to appreciate the technical complexities of what the students had produced, the innovative algorithms and efficient data structures that they had used, etc. The away team's natural emphasis on users and UX meant that the technical and user-centric aspects of the projects seemed not to be joined up in the eyes of the students. In the next iteration of DRP we took on board feedback from the students and worked to develop a single unified assessment scheme. This helped to remove the sense of "us vs them" (computer scientists vs service design specialists) that had pervaded our earlier iterations. We also paired up the instructors, so that at each point of contact with a student group – for example an end-of-iteration demo – both instructors were present and would discuss, assess and provide feedback to the team together.

At the beginning of each iteration we now give an overview lecture, reflecting on good things that we have seen teams doing during the previous iteration, and highlighting things to focus on in the upcoming iteration. We have made a conscious effort to ensure that instructors from both disciplines are present for all of these lectures, and that each speaks for a roughly equal amount of time. Presenting a common front to the students in this way seems to be effective in overcoming the home-team advantage. These joint sessions help the instructors to make sure they are on the same page, clear about what has been communicated in terms of expectations for the students that iteration, and also to speak with a common voice, using common terminology. A key challenge has been to get everyone involved in running these projects to speak the same language, so that there is one clear message at each point of contact, rather than two.

V. Course Design

The DRP course combines service design techniques with agile development and delivery of web applications to enable students to create digital software that addresses challenges faced by real people in a particular context. The course introduces students to design skills and approaches, as well as technical tools, the objective being to help teams to design, implement and evaluate innovative software that creates meaningful experiences for people in their everyday lives.

A. Prior experience

The DRP course is undertaken at the end of the students' second year of study. In order to pass the first year students are required to demonstrate proficiency in programming across a range of programming languages and paradigms. There is also a short, formative, introductory course on web programming. The second year reinforces this with a course that covers topics such as design patterns, testing and deployment. There is also a coordinated laboratory programme that involves students working in small groups on a number of practical and deeply technical projects. By the time they begin the DRP project they are accomplished programmers but have little or no experience of software development aimed at real users.

B. Technical Constraints

One of our universal objectives is to allow students to explore new technologies (particularly things like web frameworks) and to practise learning these for themselves from official documentation and tutorials that they find online. They are given a few starting points and suggestions, but given the rapidly changing world of web technologies, the expectation is that they should be able to discover, understand and use previously unseen technology without assistance. Rather than attempting to teach such technology, the role of the course tutors is to help students with more general tasks, such as how to break down problems, identify components where they might search for an existing third-party library rather than developing a feature from scratch, or navigate the wealth of online content to find a solution to their particular problem.

Although we give a lot of freedom in terms of specific technology choices, we do place a few constraints on the overall technical architecture of the system that the student teams develop, the objective being to ensure that everyone develops a system of the same basic "shape". This ensures a reasonably level playing field in terms of the technical complexity of the projects, and the approximate amount of effort it should require for teams to get some basic features working. It also simplifies the assessment. The constraints that we put in place are:

- The product should be a web application, or a mobile application with a back-end server.
- It must use a database for persistence.
- The product must support multiple users in some way, and interaction between them.
- The back-end server component must be deployed in production somewhere, not run on a local machine.

Mandating that the product must be some form of web or mobile app implies that there should be a reasonable amount of user interface design work (and implementation) carried out as part of the project. This gives an immediate connection with the human-centred design angle, as the user interface is naturally where users interact with the service, and is the area that will most easily elicit feedback. It also allows tutors to bring in considerations of accessibility, responsiveness to different screen sizes, and organisation of information.

Requiring a database steers teams in the direction that they should develop an application providing a service that users will interact with over time. The user interaction required should not be "one-shot", as in a calculator or currency converter for example, but something involving a richer user journey where the service design aspects become more important. Originally this constraint was imposed in order to ensure that students practised their database and SQL skills as part of this project. However, it has a much more nuanced effect on the types of applications that are produced. It would have been hard to write a clear marking scheme that judged the sophistication of the user interaction, but imposing a binary technical constraint such as this is something that is easy to explain and easy to check.

Building on this, we state that the application must be multi-user in some fashion. This usually necessitates providing user accounts, and some kind of login or other authentication process. This is a useful technical hurdle for the students to overcome, but more interesting is how their service supports the interaction between different users, including aspects of confidentiality, privacy and other ethical concerns.

The final constraint is that when teams demonstrate their applications they have to do so "in production". We do not allow demos running locally on developer laptops, which has a number of benefits. Firstly, requiring features to be deployed for demonstration requires them to be integrated. One developer cannot demo the feature that they have been working on alone without integrating it with the rest of the team's code. This encourages the practice of continuous integration [6] (regardless of whether or not the team has set up tooling to support this with automated builds and testing). Secondly, providing a version that runs in production on a public URL increases the quality and quantity of user feedback that teams can gather, as users can try the service on their own devices, at a time convenient to them – not just on the developer's laptop for a few minutes at the time that they come to demo it. Lastly, deploying to a production environment reinforces the use of tools like continuous build and deployment pipelines and cloud platforms, commonly badged as *DevOps*.

C. DevOps

One of the over-arching principles of our curriculum design and evolution is to allow topics to filter down from final year, specialist, elective modules, to core modules in earlier years as they become more mainstream – particularly with reference to industrial practice. We believe in threading skills through the curriculum, rather than isolating them in specific modules. For example, use of version control is something that we introduce on day one, and then students use it practically every day for the remainder of their programme. We aim for the same with other tools and techniques, and recently this has led to us embedding DevOps practices at various places in our undergraduate curriculum.

Rather than introducing the technical practices associated with DevOps [7] in a dedicated module, we have woven a thread through several different modules and projects, emphasising the underpinning nature of these practices. Wherever a software development project is undertaken as part of the degree, particularly where it is tackled as a team, we want the students to be able to draw on appropriate technical practices to support them in producing high quality work. We want students to experience first-hand the benefit of putting these tools and practices into action, and to give them a set of tools that they can, and will, use in future projects, not because we told them to, but because they know they will help them produce better results [8].

In earlier iterations of DRP, we tried to teach the students about concepts and tooling to support things like automated builds and deployment pipelines at the same time as they were developing the core of their projects. However, the result was information overload, with the effect that many teams left the construction of their pipelines until late in the projects, focusing instead on building their early prototypes where they felt they could make more progress more quickly.

Moving the introduction to DevOps earlier in the year had a transformational effect on the DRP projects. Now, by the time students embark on DRP they are familiar with a range of tools and their usage and are able to set up things like a build and deployment pipeline relatively quickly and easily. By deploying at the start of the project they are now able to lean on their pipeline throughout, ideally making several releases per day, reliably, repeatably, and with little overhead. This in turn helps them to get new versions out more quickly, and to elicit more feedback. Anecdotally, students have much more of an appreciation for the power of the pipeline when it fades into the background and becomes a routine day-to-day tool that promotes an iterative way of working, rather than a major technical hurdle to overcome.

As with programming languages and frameworks, students are allowed to make their own technology choices when it comes to infrastructure and deployment. They can either use systems that are provided by the university, that they should have used before, and which we are able to support, or they can take the opportunity to explore and look further afield. For example, they might choose to use the university's internal Git hosting for their version control, or they might choose GitHub or BitBucket if it suits them better. They might choose to deploy their application to a VM provisioned on the university private cloud (for which we can provide support), or they might choose to use a public cloud provider. Platform-as-a-Service offerings like Heroku are popular (we include use of Heroku for deployment in our guided DevOps labs), but some teams choose to use other cloud providers, often taking advantage of the free usage tiers that the vendors offer. A database-backed webapp or a backend for a mobile service will normally not incur any financial cost to deploy at the scale of these university projects, so teams can explore, but at the same time, infrastructure is not the heart of this project, so what we really encourage teams to do is to put together a set of tools that will work for them quickly and easily, so they can concentrate on delivering software and iterating rapidly. We want the DevOps work to feel like a means to an end, not an end in itself.

D. Walking Skeleton and Vertical Slices

Throughout the project we encourage teams to follow the principle of *vertical slicing*. Instead of dividing work by technological layers, we encourage a vertical slice through the application, adding just enough in each layer to implement a (small) product increment – just enough to get something in front of users to facilitate feedback.

We believe this is an important lesson as it seems to be quite natural for teams to divide work horizontally. If they divide work so that a different person works on each layer (back-end, front-end, database...) in order to complete a feature, then they introduce an integration and synchronisation problem. Before the feature can be demonstrated, everyone needs to complete their part, integrate the code, and have the APIs between the layers work together correctly. If the front-end work gets delayed, and only the back-end work is completed, there is nothing for the user to see or try. In terms of getting new user feedback, there is no value to the work.

There may also be waste. Spending time perfecting the database schema to store the data for a feature which, on contact with the user, turns out to be completely unwanted, is a waste of time and effort. Slicing vertically means starting at the front-end, and then building just enough of the back-end to support a version of the feature that can be demonstrated and trialled. This might mean hard-coding data rather than pulling it from an API, or using a file for storage rather than a database – whatever is the minimal reasonable solution to get to the point where someone can try the feature and give feedback.

Integration risks are reduced by building and deploying a *walking skeleton* early in the project. This pattern was first characterised by Alastair Cockburn, who describes it as: "a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components." [9].

As the students will often be using technology that is new to them, and connecting several components in a (small) distributed system, we encourage them to build a walking skeleton that connects their front-end, their back-end, their database, and any other key components that they plan to use (for example external web service APIs). The idea is to ensure that these can be made to work together successfully, even if in a very simple way. This mitigates the risk of discovering integration problems late on, and gives a frame for the team to build further vertical slices on top of.

E. Example Projects

Over the years, in general, the teams with the best projects (who gained the highest grades) have typically found a theme or area of focus that they are personally passionate about and where they have access to a range of relevant stakeholders.

A **typical project** is often based on current experiences in students' lives, such as living in rented accommodation with other students and related issues (e.g. payment of bills and sharing the household chores); helping students make the right choice of university; improving the running of university societies; creating better systems to organise sporting events; finding students to collaborate with on projects etc.

These projects can be successful when teams engage with a wide range of potential users. In the case of helping housemates avoid issues relating to household chores and payment of bills, a team might set out to engage with 2-3 households to gauge housemates' expectations, personalities and how things currently work. This may also involve exploring a range of apps and online platforms that students use in general to run and enjoy their lives, before creating an *opportunity statement* (see Section VI-A) from which they can start prototyping some early concepts.

The danger with theses types of projects comes when students are so familiar with the issues they are currently facing, that they do not engage with a wider group and instead develop their own ideas using a biased approach to prototyping based solely on their own ideas and opinions. They are very certain of what they want to build, and proceed on the path towards their envisioned final product. This often manifests in the project advancing through a series of "iterations" that simply create and build the features that the students planned from the start, and the only "responses to feedback" are the changing of colours, or the position of components in the interface, with no clear overall value added. Although these projects set out with good intentions, and the domain seems appropriate, the proximity of the students to the problem can often limit the degree of discovery and adaptation involved in creating the product.

In comparison, **a project that was very successful**, was with a team who initially investigated the United Nations Sustainable Development Goals with the aim of finding a topic they were passionate about and felt was important. They chose mental wellbeing as an area to explore, recognising the potential at the university by speaking to peers and members of staff involved in providing services in this area.

Having engaged with a large number of students through a range of design methods, as well as members of university staff such as academic tutors and student support services, the team created a rich bank of powerful and visual data that helped them generate a range of ideas that they went on to test with their stakeholders.

The team used such a rigorous and collaborative approach that they created a project that university was keen to take further. The project aimed to recruit second year students as mentors who would be matched with first year students in the following year and who would help support new students through the difficult first year of university. Mentors would give advice based on their own experience as well as directing their mentees to the professional services on campus.

The team presented the idea in a compelling way, not only through simple visual language and evidence, but by creating a video of a university news programme that brought the idea to life, highlighting how they planned to measure success, e.g. by tracking how many students engaged with the proposition and then also how it impacted on individual students.

Throughout the project, they continually adapted their ideas and their implementation as they learned more about the problem and how their users wanted to interact with their system. Although they made quite radical changes to their design through successive iterations, this was not a weakness, instead it was a sign of iterating, responding to feedback and adapting the product to deliver more value.

A **project that did not work so well** involved a team looking at the important issue of a student needing to earn money to support their studies. However, in terms of service design, the team did not explore the wider context of this issue and instead, created a system that would allow university students to earn money by carrying out chores and other tasks for their local community, with an aim of supporting those in particular need such as older members of the community that did not necessarily have family support.

There were two main difficulties that prevented this scenario from working as a project that met the learning objectives and also delivered a meaningful product. Firstly, the team was not able to find a large number of potential users that fit into this scenario, and so it is not clear that their product direction was centred on a real need, or a real opportunity for value creation. The second problem was that a key component of the product involved the exchange of money. Integrating real financial transactions into a software product is difficult, particularly at the prototyping stage. Users will not put their money into something they feel is incomplete or potentially insecure. But, testing the product without real money changing hands is not representative of the real usage and value proposition, so it is very hard to gather realistic feedback.

We would therefore steer groups away from products and usage scenarios that centre around financial transactions, as (at least within the scope of a university project) it is very hard to test them with users in a meaningful way.

VI. DRP IMPLEMENTATION

Our DRP project forms the culmination of the second year undergraduate programme in Computer Science. The project involves around 200 students divided into around 50 teams. After completing their taught modules and exams, the students finish the year by working on this project full-time for four weeks. There then follows a final fifth week when each group presents their work and final assessments are completed. Figure 2 gives an overview of the timeline.

A few weeks before the full-time period begins, students are introduced to the projects and are given some preliminary lectures on Human-Centred Design principles, the

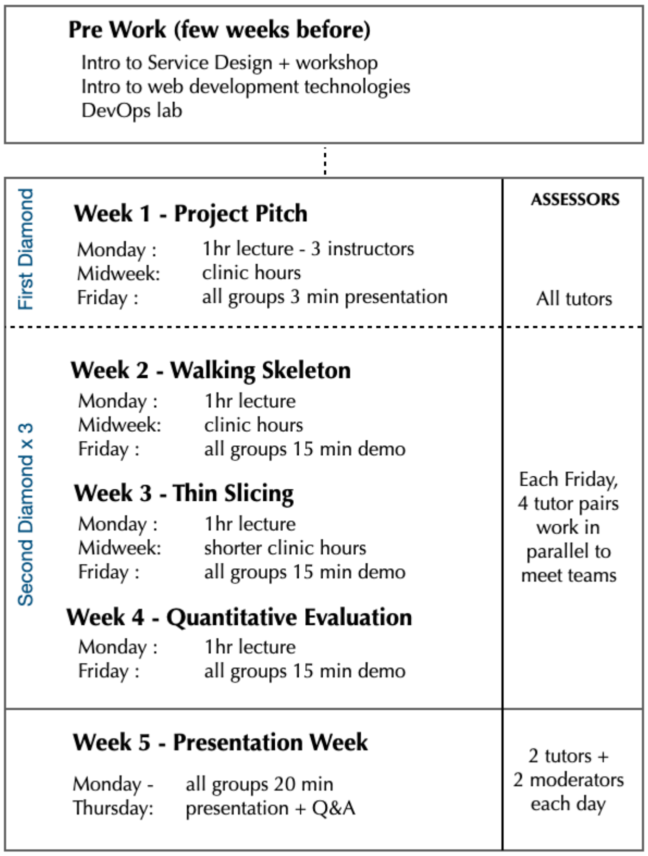
Pre Work (few weeks before)
Intro to Service Design + workshop
Intro to web development technologies
DevOps lab
First Diamond
Week 1 - Project Pitch
Monday : 1hr lecture - 3 instructors
Midweek: clinic hours
Friday : all groups 3 min presentation
ASSESSORS
All tutors
Second Diamond x 3
Week 2 - Walking Skeleton
Monday : 1hr lecture
Midweek: clinic hours
Friday : all groups 15 min demo
Week 3 - Thin Slicing
Monday : 1hr lecture
Midweek: shorter clinic hours
Friday : all groups 15 min demo
Week 4 - Quantitative Evaluation
Monday : 1hr lecture
Friday : all groups 15 min demo
Each Friday, 4 tutor pairs work in parallel to meet teams
Week 5 - Presentation Week
Monday - Thursday: all groups 20 min presentation + Q&A
2 tutors + 2 moderators each day

Fig. 2. DRP project and assessment timeline

Double Diamond process and relevant development technologies/frameworks. The HCD concepts are cemented via a hands-on workshop. Students are also presented with all the assessment criteria for the project at this time.

We have found that doing a lot of this groundwork ahead of the main part of the project means that students have more cognitive space to gain a proper understanding of the design techniques and the framework in which they are expected to operate, how they will be assessed, and the technology options available to them. They have a few weeks to let these ideas sink in, and to finish off the assessments for their other modules, before embarking on the main part of the project where they will work with their team full-time.

The four week full-time period is divided into four one-week iterations. During the first week activities focus on the first diamond, conducting research and refining the problem, then weeks two through four move to the second diamond, iteratively designing and delivering the solution, while constantly seeking and adapting to feedback, with the aim of gradually converging to a final product. Each week has a particular theme for the students to focus on, each with slightly different assessment criteria. We begin each week with a lecture setting the scene, highlighting the theme and the assessment points. During earlier weeks we hold midweek clinic hours to support the teams, but demand for these diminishes in later weeks.

A. *Week 1 - The First Diamond*

In the first week, students dedicate their time to identifying a problem that will become the focus of their project (the first diamond in Figure 1). This involves researching candidate challenge areas and gathering data on the needs of relevant users, e.g. by assembling numerical data (quantitative) or through user interviews (qualitative). Students are not expected to do any coding in the first week; the focus is on identifying a problem and a set of user needs, although early ideas may emerge from this towards a digital touchpoint.

Assessment is carried out at the end of the week via a 3-minute elevator pitch presentation which summarises the problem, audience, supporting evidence (including insights gained through research and/or user interviews) and an *opportunity statement*, which typically begins "How might we...".

An example opportunity statement from a recent project read: "How might we enable food charity project managers to strengthen and efficiently organise their volunteers in order to create a vibrant and reliable volunteer network whilst reaching increasing numbers of people in need".

Students often find it easier to write a *challenge statement* describing the general problem area – e.g. in this case "Hundreds of thousands of tonnes of good food is wasted by the food industry every year ... food charity X works to support people in need, but often struggles to manage supply and demand of volunteers". However, we push teams to re-form this as an opportunity statement, as this typically identifies the users, the need, and ways in which the success of the final deliverable might be measured.

B. *Weeks 2-4 - The Second Diamond*

During the second week students are required to implement and deploy a simple walking skeleton (Section V-D) that will form the backbone for subsequent development and user testing. The groups then develop the core features of their products through thin vertical slices, adding features incrementally, working across the software stack. Having the walking skeleton deployed to production allows them to deliver each of their features in a form where they can test them with their users and gather feedback. Students follow this second diamond repeatedly over three, or possibly more, iterations. In each of these they build new features, gather feedback, and adapt their designs and plans accordingly. The assessment structure requires one iteration per week, with a new product increment to demo each Friday, but we encourage students to follow a tighter iteration cycle if they can, adding thin slices and testing with users multiple times within each weekly cycle.

Within weeks 2-4 we have three different weekly themes. The first is to develop the walking skeleton and a few core features. The second theme is around making meaningful progress on the product, getting feedback and changing direction as needed to align closely with the user's needs. The third is to define one or more metrics that teams can measure and use to optimise particular aspects of their user experience (for example the time or number of clicks needed to achieve a certain task). Optimising something that is the wrong feature is a waste of time, so we encourage this quantitative work only later in the project, once the teams have gathered enough qualitative feedback to feel confident that their overall product direction is correct.

C. Assessment

At the end of each week we have a checkpoint where the student groups present and discuss their work with the assessment team. As we have a large class we have had to devise ways to make these meetings short, while still giving enough time for the team to feel heard, to mark accurately, and to provide some useful feedback.

The elevator pitches at the end of week 1 are seen by all assessors with brief oral feedback given directly afterwards, but in weeks 2–4 each project is assessed by a pair of assessors, one from each of the two disciplines, in a 20 minute meeting. The same pair follows the group over the remainder of the project and this is an important aspect of the assessment: we previously learnt that changing assessors from one week to the next leads to inconsistent feedback and a lack of trust both in the process and the way the projects are assessed.

At the weekly assessment meetings, the assessors use a checklist to assign marks of 0, 1 or 2 in each of a number of categories. This simple scheme helps to make the assessment quick and consistent across pairs. The students also felt happier when the criteria were more concrete and not (at least in their opinion) open to interpretation by the assessors. It seems to be another natural trait of computer scientists that they are concrete people, used to problems with right or wrong answers, so being assessed on subjective qualities like "quality of visual design" or "ease of use" did not sit well with them. It worked much better to have criteria such as "had no evidence of feedback from users" / "had 1-2 video clips of user test sessions" / "had 5+ video clips with a variety of users".

In addition to the checklist matrix for assigning marks, each week there is also a pre-prepared list of follow-up tasks that the assessors can choose from to assign to the teams as actionable feedback. These task lists have been accumulated in the light of experience and are designed to make feedback concrete and useful from the students' perspective whilst being lightweight to administer from the assessor's perspective. For example, for the walking skeleton milestone some of the suggested follow-up tasks are: "Go to [more / different] real people with your mock-up", "Solve technical issues with the deployment pipeline" or "Decide how to prototype the next stage based on learning about [usability / desirability / functionality]". The assessment pair keeps a copy of the checklist showing which tasks were assigned so that they can pick up on them the following week.

Note that we do not separately assess the extent to which students are adhering to agile development practices, because the schedule of work, and the weekly assessment, essentially enforces the practices we want the students to learn. For example, we do not prescribe a schedule of meetings and ceremonies and then check to see whether students have carried them out, instead, we leave it to students to adapt their ways of working as they think best to help them hit the assessment targets and by extension to deliver a better product.

For the final assessment (which happens in a fifth week) each tutor pair assesses their groups over the course of a day. Although these pair assessments could all run in parallel we have found it useful to have an additional moderator from each discipline sit in on all the presentations in order to ensure consistency of assessment and feedback. This may not be possible to maintain if we need to scale to a much larger class in future.

VII. Related Work

With the near ubiquity of agile methods among modern software development teams, it is no surprise that the majority of universities teaching Computer Science and Software Engineering cover agile methods as part of their programmes [10], and that this is often associated with a practical team project. Agile methods are often taught or applied in capstone courses [11], [12] and particularly using Scrum [13], [14], although some courses have started to introduce other agile methods such as Kanban [15] as industry trends change. De Souza et al [13] report observations similar to ours that students felt that the Scrum ceremonies were an overhead, and that they were often dropped in favour of more time on development: "... a lack of commitment with Scrum practices was reported due to the amount of overwork".

In this experience report it is not our aim to perform a rigorous literature survey comparing courses at other universities, merely to present our own experiences and our reflections upon them. However, from programmes that we are aware of, and through conversations with colleagues, it is apparent that the teaching and assessment of agile methods is often focused on processes and tools [16], [17]. We have taken a different approach, for the reasons described in this paper.

The project course that we have developed combines software engineering practices with elements of service design. Service Design is not a new concept and has been recognised in recent decades by academics and practitioners as a discipline and approach that can transform the quality and value of customer, employee and citizen experiences [18]. Although there is no exact definition, there is broad agreement that Service Design entails an iterative systematic, human-centred, holistic, creative, and iterative approach to creating new service-oriented systems [3], [4].

Service Design is closely associated with the Design Thinking process of problem solving – "a methodology that imbues the full spectrum of innovation activities with a human-centred design ethos" [19]. A large number of university-level courses are now being offered in the areas of Service Design and Design Thinking. Linking these with computer science can prove particularly effective, because the prototyping and feedback cycles involved can be realised through developing software, which is easily malleable. A number of Design Thinking case studies have been reported in computing-related areas such as games programming [20] and more general aspects of software development [21], [22]. There are also interesting parallels that can be drawn between Design Thinking and the more structured processes associated with HCI [23], both of which are inherently iterative and driven by interaction with, and feedback from, users.

A critique of the focus on ceremony and process in industrial software projects is given by Mancuso et al in the work of the Software Craftsmanship movement [24]. Reflecting on experiences from a wide range of commercial projects, many of which failed to deliver value sustainably, the idea of Software Craftsmanship is to focus on the technical quality of the software being produced. Key concerns in Software Craftsmanship are the degree to which the software is extensible, testable and maintainable, with the view that these technical concerns support agility, and that agile processes alone will not deliver value without supporting engineering practices.

Perhaps closer to our focus on customer needs is the Lean Startup movement and the work of Ries et al [25] which has seen widespread adoption in Silicon Valley and beyond. Here the focus is on continuous learning and feedback based on collecting data about customers. Rather than deep engineering efforts, the minimum viable software is produced at each iteration, just enough to elicit meaningful feedback and learning. As Ries writes “Success is not delivering a feature; success is learning how to solve the customer’s problem.”.

VIII. Reflections and Recommendations

We have found interdisciplinary collaboration to be very valuable to software engineering projects. In our case, the collaboration was with experts in the area of Service Design and user research, and those experts happened to come from another institution nearby. However, we believe that similarly fruitful collaborations could be formed between different departments or even different groups within the same institution. The main value came from adding a different angle, that lifted the students away from a purely technical focus in their work.

When software engineering students talk to software engineering staff about software engineering projects, it is only natural that their focus tends towards technology. However, we know that for software development work to be useful and valuable, we need to address the needs of the user, not to develop technology for its own sake. Indeed, the development of complex technological solutions often creates a liability. As Daniel Terhorst-North puts it “software is like surgery; no one wants surgery, they just want to be well”[2]. Our recommendation is to construct a project where the focus is outside of software engineering, but where software and good software engineering practices can provide a means to an end. Our observation is that this helps students to see the value in the practices they are applying, and to appreciate them in a wider context. Interaction with experts from another discipline also naturally provides students with an additional source of knowledge and skills which helps to broaden their education.

We discovered over many years that making such a collaboration work is difficult, particularly getting to the point where students view the two parties as equal partners and not opposing teams. There may often be a need to overcome a home-field advantage to ensure that both teams’ input is respected. Alignment of message and equal emphasis is important. We have a number of practical recommendations to help with this. Firstly, whenever there is contact between staff and students, or there is content delivered, the staff should appear as a team, with representation from each discipline. We also recommend using a common style and format for slides and written materials, again, to emphasise unity between the disciplines, with no distinct boundaries. When assessment is done, it should be done by a pair or team of assessors, again with representatives of each discipline. Feedback should be given directly, so that both assessors are part of the same conversation with the students. There is no “we heard X from them, but now you’re telling us something different...” which was something that we heard a lot when different assessors met students separately, and often gave (seemingly) inconsistent feedback. There should be a single, unified set of assessment criteria at each point, not one set that person X is looking for and a different set that person Y is looking for.

A limitation of this approach is the time taken to develop a shared vocabulary and message amongst the two disciplines. The teams of instructors need to unify around the themes being taught so that they are aligned and reinforce one another. In the example presented here this took a few iterations of the course to establish and refine, and so shorter-term collaborations, or high churn in personnel, may make it difficult.

Another practical limitation is trying to align the scheduling of the course so that it fits with the timetables of both departments or institutions. In our collaboration this has been possible, but has meant planning a long time in advance in order to make sure that staff are consistently available to attend the student contact sessions and create a consistent experience for the teams across the weeks.

One of the things that we have found to work well is scheduling these projects at a time in the academic year where students can work on them full time, with no other timetabled modules running alongside them. The aim is to encourage team collaboration and broad band communication. Providing a work schedule where team members are able to work together synchronously, ideally in the same physical environment (and if not, then synchronously online via video conference) supports this. Hopefully this also simulates a commercial work environment (at least the one envisioned by Beck and Cunningham in the original book on Extreme Programming [26]), where a team gets to sit together and work together for most, if not all, of their working week.

We have run different types of projects in other courses where students work in teams, but where each student has a varied and individual schedule of other classes to attend alongside working on their project, which often leads to a fragmented team. The intensity of the work and the collaboration that we see in the full time projects means that even with a much shorter overall duration, and perhaps with a smaller team, we see that teams get more done, and within

[2]This quote from a talk is recorded in Graham Russell’s blog post: https://blog.ham1.co.uk/2016/04/03/software-engineering-as-surgery-part-i/

each iteration they are able to develop a significant increment to their product and get meaningful feedback. Overall our recommendation would be to favour a full time project over part time, even if the total duration is shorter, as long as there is enough time for multiple iterations to be completed.

Modern software engineering is a broad subject, with many different aspects and topics. Our experience has been that trying to teach too many of these in a single module – and perhaps asking students to learn and apply them all in a single project – has led to students feeling overwhelmed, and gaining only a shallow understanding of each. We have found it more effective to separate out some of the topics (in our case things like core programming skills, automated testing, DevOps) and to allow students to get a firm grasp of these before coming into a team project. This allows the students to apply technical practices in the project context without expending too much cognitive effort on the processes and tools. This leaves them mental space to concentrate on the interactions with their users, and to plan and prioritise the features to deliver to maximise value, leaning on the tools to support them in doing this effectively. If there is no option in the curriculum to separate out the learning of skills around topics like DevOps into separate units, we would advise teaching these tools and processes right at the start of a software engineering project, and using a small amount of assessment credit as a lever to get students to put them in place early on. This helps students feel the benefit of using this infrastructure throughout a project where they deliver iteratively.

When teaching agile methods it can be tempting to base assessment on whether students are carrying out a process "correctly" by checking whether various activities have been completed. In order to scale assessment, particularly when having to teach larger classes, checking off whether various processes and tools are being used (for example meetings, backlogs or pipelines) can seem like a good direction. However, what we have seen is that this can lead to students paying lip service to the process, carrying out each part in the minimal way necessary in order to collect the marks, and not really understanding the purpose or the principles behind these activities. Alternatively, we have seen teams going all in on a particular process to do it "by the book", but never standing back to inspect and adapt. Our advice is that concentrating on principles (e.g. delivering iteratively, responding to user feedback, adapting work processes over successive iterations etc) and setting up an assessment framework that incentivises these things leads to deeper lessons being learnt, and a greater understanding of the principles of agility.

Acknowledgments

We would like to thank the wider team of tutors from both Imperial College London and the Royal College of Art who have supported and mentored students through these projects over the years, and whose feedback and willingness to adapt has helped to refine the course to reach its current state.

References

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," 2001. [Online]. Available: http://www.agilemanifesto.org/

[2] E. Papatheocharous and A. S. Andreou, "Empirical Evidence and State of Practice of Software Agile Teams," *J. Softw. Evol. Process*, vol. 26, no. 9, p. 855–866, sep 2014. [Online]. Available: https://doi.org/10.1002/smr.1664

[3] D. Sangiorgi and A. Meroni, *Design for Services*. Taylor & Francis Ltd, 01 2011.

[4] R. M. Saco and A. P. Goncalves, "Service Design: An Appraisal," *Design Management Review*, vol. 19, no. 1, p. 10, 2008.

[5] Design Council, "Framework for Innovation: Design Council's Evolved Double Diamond," May 2019. [Online]. Available: https://www.designcouncil.org.uk/our-work/skills-learning/tools-frameworks/framework-for-innovation-design-councils-evolved-double-diamond/

[6] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, ser. Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2007. [Online]. Available: http://my.safaribooksonline.com/9780321336385

[7] N. Forsgren, D. Smith, J. Humble, and J. Frazelle, "2021 Accelerate State of DevOps Report," Google, Tech. Rep., 2021. [Online]. Available: http://cloud.google.com/devops/state-of-devops/

[8] R. Chatley and I. Procaccini, "Threading DevOps Practices through a University Software Engineering Programme," in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEET)*, 2020, pp. 1–5.

[9] A. Cockburn, "Walking Skeleton," Aug 2008. [Online]. Available: https://wiki.c2.com/?WalkingSkeleton

[10] D. F. Rico and H. H. Sayani, "Use of Agile Methods in Software Engineering Education," in *2009 Agile Conference*, 2009, pp. 174–179.

[11] M. Persson, I. Kruzela, K. Allder, O. Johansson, and P. Johansson, "On the use of Scrum in Project Driven Higher Education," in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, 2011.

[12] V. Mahnic, "A Capstone Course on Agile Software Development using Scrum," *IEEE Transactions on Education*, vol. 55, 2011.

[13] R. T. de Souza, S. D. Zorzo, and D. A. da Silva, "Evaluating Capstone Project through Flexible and Collaborative use of Scrum framework," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–7.

[14] M. Paasivaara, J. Vanhanen, V. T. Heikkilä, C. Lassenius, J. Itkonen, and E. Laukkanen, "Do High and Low Performing Student Teams Use Scrum Differently in Capstone Projects?" in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 2017, pp. 146–149.

[15] C. Matthies, "Scrum2kanban: Integrating Kanban and Scrum in a University Software Engineering Capstone Course," in *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*, ser. SEEM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 48–55. [Online]. Available: https://doi.org/10.1145/3194779.3194784

[16] C. Anslow and F. Maurer, "An Experience Report at Teaching a Group Based Agile Software Development Project Course," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 500–505. [Online]. Available: https://doi.org/10.1145/2676723.2677284

[17] H. H. Løvold, Y. Lindsjørn, and V. Stray, "Forming and Assessing Student Teams in Software Engineering Courses," in *Agile Processes in Software Engineering and Extreme Programming – Workshops*, M. Paasivaara and P. Kruchten, Eds. Springer, 2020, pp. 298–306.

[18] Q. Sun and C. Runcie, "Is Service Design in Demand?" *Design Management Journal*, vol. 11, pp. 67–78, 10 2016.

[19] T. Brown *et al.*, "Design Thinking," *Harvard Business Review*, vol. 86, no. 6, p. 84, 2008.

[20] E. R. Hayes and I. A. Games, "Making Computer Games and Design Thinking: A Review of Current Software and Strategies," *Games and Culture*, vol. 3, no. 3-4, pp. 309–332, 2008.

[21] "Design Thinking Integrated in Agile Software Development: A Systematic Literature Review," *Procedia Computer Science*, vol. 138, pp. 775–782, 2018.

[22] A. Wölbling, K. Krämer, C. N. Buss, K. Dribbisch, P. LoBue, and A. Taherivand, *Design Thinking: An Innovative Concept for Developing User-Centered Software*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 121–136. [Online]. Available: https://doi.org/10.1007/978-3-642-31371-4_7

[23] H. Park and S. McKilligan, "A Systematic Literature Review for Human-Computer Interaction and Design Thinking Process Integration," in *International Conference of Design, User Experience, and Usability*. Springer, 2018, pp. 725–740.

[24] S. Mancuso, *The Software Craftsman: Professionalism, Pragmatism, Pride*, ser. Robert C. Martin Series.

[25] E. Ries, *The Lean Startup : How Constant Innovation Creates Radically Successful Businesses*. London; New York: Portfolio Penguin, 2011.

[26] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley Publishing Company, 1999.