2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)

Dual-Track Agile in Software Engineering Education

Cécile Péraire
Carnegie Mellon University (Silicon Valley Campus)
Electrical and Computer Engineering
Moffett Field, CA 94035, USA
Email: cecile.peraire@sv.cmu.edu

***Abstract*—The fields of Software Engineering and Human-Computer Interaction have traditionally evolved in parallel, with little cross-pollination, both in industry and academia. However, effectively delivering software products offering superior user experiences requires a tight collaboration between professionals from both fields. In recent years, some approaches combining the two perspectives have been proposed in industry, including dual-track agile software development. Yet, very few courses cover those integrated approaches in academia, and it appears that no publication so far has reported the existence of a scholarly course covering dual-track agile. This paper introduces a course that addresses the divide between Software Engineering and Human-Computer Interaction through an integrated approach to requirements engineering and interaction design, in the context of dual-track agile. The course design combines traditional and flipped-classroom delivery, together with project-based learning. During the course project, students learn to design and implement software systems that address real problems and satisfy real stakeholders' needs by being useful, usable, and enjoyable to use. This paper documents the author's experience designing and teaching the course over the past four years. It aims to convince, inspire, and enable others to teach similar courses, bringing interaction design to the forefront of agile software development.**

***Index Terms*—dual-track agile, continuous product discovery and delivery, user-centered design, design thinking, requirements engineering, course design, software engineering education**

I. Introduction

The fields of Software Engineering (SE) and Human-Computer Interaction (HCI) have traditionally evolved in parallel, with little cross-pollination, both in industry and academia [1]. On the one hand, SE is primarily concerned with building software products effectively, in order to enable the continuous delivery of *useful features* to customers. On the other hand, HCI focuses on designing products offering quality user experiences, encompassing *usefulness*, *usability*, and *emotional impact*, in order to fully satisfy the users' needs [2]. Therefore, without a tight collaboration between professionals from both fields, software engineers might deliver feature-rich products that do not satisfy the needs of the users, while interaction designers might create excellent designs not reflected in the actual software product.

As a result, and for decades, users have endured painful hours behind their personal computers, somehow convinced that the struggle was a normal part of the user experience [3]. This perception has started to change, however, through a series of design innovations that have introduced many users to superior experiences for the first time, including Apple's release of the iPod and iTunes in 2001, and the iPhone in 2007 [4]. Since then, users' expectations have increased, and user experience has become a key competitive factor in the industry.

Responding to market demand, some companies have started to combine the perspectives of both SE and HCI communities into integrated software development approaches [5]. Most of these approaches share similar underlying principles, including parallel interwoven tracks of interaction design and software development, as best exemplified by dual-track agile software development [6]. Unfortunately, the disconnect between SE and HCI remains prevalent in academia, where very few courses cover both disciplines in an integrated fashion. To the best of the author's knowledge, no publication has reported the existence of a scholarly course covering dual-track agile. Furthermore, the way most software engineering programs teach design may be inadequate [7] because core design topics are omitted.

This paper introduces a course, called *Software Requirements and Interaction Design (SRID)*, that addresses the divide between SE and HCI by exposing students to an integrated approach to Requirements Engineering (RE) and Interaction Design (IxD), in the context of dual-track agile. The dual-track agile process model proposed in the course is consistent with the dual-track agile process observed by the author and her colleagues at Pivotal Software in the context of an extensive participant-observation study [8]. The course adopts an agile approach to RE, which embraces changes, and where the term *requirement* denotes a property of the solution that is desired by one or more stakeholders and *might* end-up being reflected in the implementation (versus a property that *must* be reflected in the implementation).

SRID is a graduate-level course in Carnegie Mellon University's Master of Science in Software Engineering program offered on the Silicon Valley campus. Students enter the course with no experience in interaction design, but have taken another course of the program [9][10] where they experienced 'traditional' agile development.

The course delivery format is based on a mixed approach combining flipped-classroom delivery [11] and traditional delivery in the context of project-based learning [12]. During

978-1-7281-1000-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-SEET.2019.00013

the course project, students learn to design and implement software systems addressing real problems, satisfying real stakeholders' needs, and offering superior user experiences. Students also get a taste of the continuous and concurrent nature of product discovery and delivery.

This paper documents the author's experience designing and teaching the course over the past four years. Section II documents the state of the art, as it relates to the connection between SE and HCI, in both industry and academia. Section III introduces the course's core process, dual-track agile. Sections IV and V describe the course design, together with the challenges involved in teaching such a course. Finally, Section VI concludes with the paper's key contributions.

II. State of the Art

While the SE community has a lot to say about *how to build* software products, it generally does not offer much guidance about determining *what to build*. The Scrum Guide [13], for instance, remains silent on the topic. While the Scrum Product Owner is in charge of deciding what product or feature to build, there are no Scrum activities to support that decision. How does the Product Owner go about deciding that a specific feature needs to be built? *"How it is done may vary widely across organizations, Scrum Teams, and individuals."*[13]. It is left to each to figure it out.

Similar to Scrum, Extreme Programming [14] does not offer practices to help the team determine what to build, except for one advanced practice called *Real Customer Involvement* (formally *On-Site Customer*) where an on-site customer writes and prioritizes user stories [15]. The goal of this practice is *"to reduce wasted effort by putting people with the needs in direct contact with the people who can fill these needs"*[14]. This practice effectively supports the goal of building a product satisfying the real customer needs. However, it falls short of guiding the customer and team in figuring out what to build.

The same applies to Large-Scale Scrum (LeSS) [16] which includes *Customer-Centric* as one of its core principles, with a goal of ensuring that everyone is connected to and cares about real customers. This is done by defining the Product Owner as a connector of customers/users and teams, rather than an intermediary, and by focusing on feature teams that are aligned with creating end-to-end customer-centric features. Feature teams, together with customers and users, refine items to be ready for future Sprints during the *Product Backlog Refinement*, hence freeing time for the Product Owner to focus on customers. Here as well, the approach falls short of guiding the Product Owner, feature teams, customers, and users in determining what to build.

Meanwhile, software requirements engineering tends to primarily present the definition of a solution as a sequence of requirements refinement steps. For instance, the requirements process presented in [17] starts with stakeholders' needs, from which to-be features are derived, then high-level requirements, then detailed requirements elaborated just-in-time (typically just before implementation). Similarly, in [15], large stories (or epics) are broken down into smaller stories, which are then further refined using acceptance tests (typically written just before implementation based on conversations between developers and stakeholders). In [18], the author defines a *requirements chain* going from business requirements, to user requirements, to functional requirements. These approaches mostly ignore, or present as secondary, the creative dimension of defining what to build.

The HCI community, unlike the traditional SE community, is fundamentally concerned about the design of user experiences for interactive products, and has proposed various models aimed at explaining the creative process of determining what product or feature to build. Examples include the Goal-Directed Design process [19], the Design Thinking Process [20], the Double-Diamond Design Process Model [21], and the Wheel Lifecycle Template [2]. These models are supported by a rich set of generally accepted user-centered practices that guide the creative process [22]. Unfortunately, these user-centered processes and practices have been developed outside of the SE community, and only few have been incorporated into popular SE methods. As a result, they are regarded as unimportant by some in the SE community [1].

The challenges involved in bridging the gap between the SE community's focus on *how to build* and the HCI community's focus on *what to build* are described in [23] together with some early solutions. In recent years, and as illustrated by a number of systematic literature reviews [5][24][25], we have witnessed the emergence of strategies aiming at integrating the perspectives of the two communities. Out of these strategies, [5] identifies five fundamental principles underlying a user-centered agile software development approach: continuous stakeholder involvement, artifact-mediated communication, iterative and incremental design and development, separate product discovery and product creation, and parallel interwoven tracks of design and development.

The idea of parallel interwoven tracks of design and development is what some call *dual-track agile* [6][26][27]. The term *dual-track agile* originates from a 2005 paper [28] presenting an effort to merge agile development and user-centered design at Alias. This effort is also presented in a 2007 paper [29] describing software development at Autodesk (formerly Alias). Both papers introduce a software process based on two tracks: An *interaction designer track* aiming at discovering what to build, and a *developer track* aiming at building software, with interaction designers staying at least one cycle ahead of developers.

A similar and recent application of dual-track agile in industry is reported in [8], which presents the results of a grounded theory study conducted at Pivotal Software. In this context, the first track typically includes user research, negotiating with stakeholders, drawing user interface mockups, and writing user stories. The second track typically involves building, testing, architecting, refactoring, deploying and maintaining the product. The two tracks run continuously and in parallel, and the whole team is involved in both tracks. However, product designers lead the first track, while engineers lead the second track. The product backlog, owned by product managers, serves as a boundary object between the two tracks.

The gap between SE and HCI remains important in academia, where very few courses cover both disciplines in an integrated manner. A few notable exceptions are presented later in this section. Furthermore, the way most software engineering programs teach design might be inadequate. A study [7] of the ACM/AIS Model Curricula for Software Engineering and Information Systems [30][31] reveals that these model curricula omit core design topics and insufficiently cover how to generate design candidates. The study explains that while we teach agile methods, we teach software *evolution* (the gradual evolution of a software object through iterations) but miss the fundamental concept of *coevolution*. *Coevolution* refers to *"developing and refining together both the formulation of a problem and ideas for a solution, with constant iteration of analysis, synthesis and evaluation processes between the two notional design 'spaces'—problem space and solution space"* [32]. An example of a course explicitly introducing students to coevolution can be found in [33]. Finally, while designers embrace divergent thinking in order to explore alternatives, agile methods follow a path along a single trajectory. Hence, while *"you might get a sub-optimal design right, you will almost never get the right design"* [34].

Divergent thinking and coevolution are the cognitive phenomena underlying design thinking. A few recent attempts at integrating design thinking in SE education have been reported in the literature. For instance, in [35], the authors describe their first experience conducting a design thinking workshop to help students generate product ideas at the beginning of a course project, and their decision to incorporate the workshop in future offerings of the course. A similar experiment is reported in [36], where graduate students applied the design thinking process over a two-week period. One of the lessons learned is that, even though design thinking seems simple at first, students need more time to assimilate the various techniques and successfully put them into practice.

A more ambitious experiment is reported in [37], which describes the authors' first experience introducing design thinking in the context of a software engineering capstone project course. Throughout the semester, students apply various interaction design practices in the context of a staged development process (planning, requirements, analysis and design, testing, maintenance, and deployment). The capstone course described in [37] shares many similarities with the course presented in this paper. A key difference, however, is that our interaction design practices are integrated within an agile software development process (versus a staged process) called dual-track agile.

III. A Road to Dual-Track Agile

The course presented in this paper starts by introducing two main characters, Remi, the Requirements Engineer (often called Product Manager), and Iris, the Interaction Designer, as well as their backgrounds and traditional ways of working.

Remi, our Requirements Engineer (or Product Manager), belongs to a software development team and is in charge of defining and managing software requirements. Iris, our Interaction Designer, belongs to a user experience team and is in charge of designing software behavior. While both Remi and Iris share the same mission, namely, *conceiving software systems addressing real problems and satisfying real user's needs*, they each have a very different way of fulfilling their mission.

Indeed, Remi and Iris have very different backgrounds and perspectives. Remi holds a degree in software engineering, where the main concern is the development of software systems. He perceives his domain as a set of complementary disciplines, including project management, requirements engineering, architecture and (technical) design, implementation, testing, and quality assurance. Iris holds a degree in Human Computer Interaction (HCI), where the main concern is the design of user experiences for interactive computing systems. She perceives her domain as a combination of form (industrial and graphic design), content (information architecture), and behavior (interaction design) [19].

A. A Requirements Engineer's Perspective

This section describes a hypothetical process that Remi, our Requirements Engineer (or Product Manager), and his software development team might follow in order to deliver a new feature or product. The process is presented from Remi's perspective, and illustrated on the left-hand side of Figure 1. It is inspired by the requirements process presented in [17] and the agile literature listed in Section II. Despite oversimplifying reality, the process aims at being reasonably representative of the way agile teams traditionally go about developing software. The process activities can be summarized as follows:

- **Elicit Requirements**. Remi starts by assembling the key stakeholders in a conference room to conduct a requirements workshop. His goal is to understand their needs and requests, and to come up with an initial list of product features and requirements. The marketing manager might be present and propose some features based on competitive products. The list is refined through discussions involving domain experts and potential users.
- **Envision Requirements**. Based on the information previously gathered, Remi creates or updates, in an ad hoc manner, a long-term vision for the product to be built, and a set of features to be potentially implemented during the next release(s). The features are broken down into a list of prioritized high-level requirements (e.g., epics). As needed, Remi researches the competitiveness of the new product ideas by conducting a competitive analysis, and a software developer demonstrates their technical feasibility with a proof of concept.
- **Elaborate Requirements**. Remi starts to elaborate the epics in priority order. Each epic is broken down into smaller user stories with acceptance criteria. Remi adds and prioritizes the user stories into the product backlog. Since the epics are elaborated just-in-time, **Elaborate**

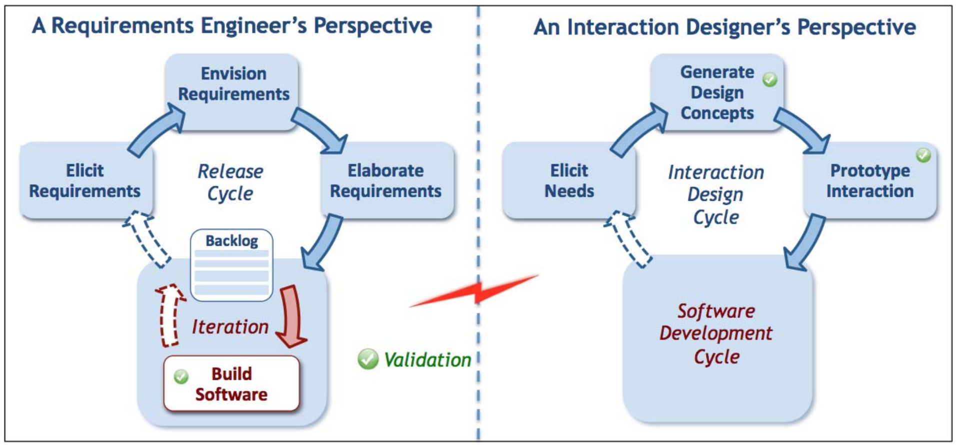


Fig. 1: Two distinct perspectives on software product development

Requirements might continue while **Build Software** is underway.

- **Build Software**. Remi presents the high-priority stories to be developed during the coming iteration to the software developers. Each developer selects one or more stories at the top of the backlog, and gets to work. Once these stories are implemented, they are reviewed by Remi and the rest of the team to verify that the working software satisfies the acceptance criteria. Potential users and other stakeholders might be invited to the review, in order to validate that the team is building the right features or product, and to discuss the direction of the product.

The product development might continue with one or more iterations until a set release date. The product development might end at that point, or move on to the next release cycle.

The scenario above illustrates a tight and effective collaboration between Remi and the developers on his team. Remi is capable of decomposing the envisioned solution into small cohesive pieces that could be built individually, one after the other, to implement the solution incrementally. Remi organizes the work for the developers ahead of time (via the product backlog), and verifies the work after the fact. This happens with little documentation and communication overhead.

Because the development process is mostly feature-driven, the resulting product is likely to be feature-rich. Would these features satisfy the user needs? Would the product be useful, usable, and enjoyable to use? Given the potential lack of user involvement, the answer may be *no*. Furthermore, Remi assumes that the stakeholders know, and are able to articulate, what they want (e.g., features), and that what they want is indeed what they need. Again, this might not be the case.

Finally, in the scenario above, only one possible solution is considered, as there are no opportunities to consider alternative ideas for the solution using divergent thinking. Furthermore, because the definition of the solution is based mostly on a sequence of requirements refinement steps, Remi and his team do not have the opportunity to imagine a solution through coevolution—the rapid cognitive oscillation between our understanding of the problem and ideas for the software product [7].

B. An Interaction Designer's Perspective

This section describes a hypothetical process that Iris, our Interaction Designer, and her user experience team might follow in order to design a new functionality or product. The process is presented from Iris' perspective, and illustrated on the right-hand side of Figure 1. It is inspired by the Wheel Lifecycle Template [2] and other interaction design models listed in Section II. Despite oversimplifying reality, the process aims at being reasonably representative of the way user experience teams traditionally go about supporting software product development. The process activities can be summarized as follows:

- **Elicit Needs**. Iris, with help from other members of her team, starts by researching the domain within which an opportunity has been identified. Once they know enough about the domain, they conduct in-person interviews of key stakeholders, including contextual inquiries [38] with potential users. Because their objective is to understand the stakeholders' goals and related needs, they purposely stay away from discussing future features or product. The team makes sense of the data gathered during the interviews by performing a contextual analysis (e.g., using affinity mapping [39]) leading to an empirical identification of the stakeholders' needs.
- **Generate Design Concepts**. Once Iris and her team have a reasonable understanding of the needs, their objective is to generate, and experiment with, alternative interaction design concepts that could satisfy these needs. To do so, they first brainstorm design ideas, often using sketches to boost creativity [40]. The team combines and synthesizes these ideas, and then showcases the best resulting concepts using storyboards [41]. These storyboards illustrate how personas [42] interact with the software solution to achieve their goal within the context of use. Finally the team conducts a few design walkthrough sessions [43]

with potential users. By walking the users through the frames of the storyboards, they validate and refine their understanding of both the domain and interaction design concepts. With these storyboards, Iris and her team *"plant and first nurture the user experience seed"* [2].
- **Prototype Interaction**. Iris illustrates the interaction design concept that resonated the most with users during the design walkthrough sessions, by creating a low-cost medium-fidelity prototype simulating the final product. She then conducts usability testing sessions with a few users for validation purposes. Cycles of refinement and validation follow until the feedback becomes inconsequential. The validated prototype is then sent to the software developers.
- **Software Development Cycle**. The software development team is now in charge of developing and delivering a software product based on the prototype. Except for potential clarification questions from developers on the prototype, Iris and her team have completed their mission for this interaction design cycle.

The product development might end at that point, or move on to the next interaction design cycle.

The scenario above illustrates how Iris and her team are capable of identifying stakeholder's needs based on data, and how they effectively involve the users throughout the design process to make sure that the solution satisfies their needs.

Iris and her team apply design thinking to generate and validate alternative design candidates. Using divergent thinking during brainstorming sessions, they generate alternative viable interaction design concepts. Through multiple rounds of coevolution (e.g., sketching, storyboarding and design walkthrough, prototyping and usability testing), they simultaneously refine their understanding of both the problem and the solution. As a result, their interaction design is likely to be innovative while satisfying the users needs.

Unfortunately, the scenario also highlights a lack of communication and collaboration between the user experience and software development teams, and a lack of actionable work items (e.g., user stories) necessary to guide the software engineers during their implementation of the proposed interaction design. As a result, it is quite possible that the proposed interaction design will never be fully (if at all) reflected in the final product.

C. An Integrated Approach: Dual-Track Agile

This section presents yet another hypothetical scenario aiming at delivering a new functionality or product. This time however, Remi, our Requirements Engineer (or Product Manager), and Iris, our Interaction Designer, are working together on the same cross-functional team, along with a few software engineers. The proposed process, illustrated in Figure 2, is not meant to represent any given company's process. However, it is consistent with the dual-track agile process that the author and her colleagues have observed at Pivotal Software in the context of an extensive participant-observation study [8][44][45][46], and other instances of dual-track agile presented in the literature, as described in Section II.

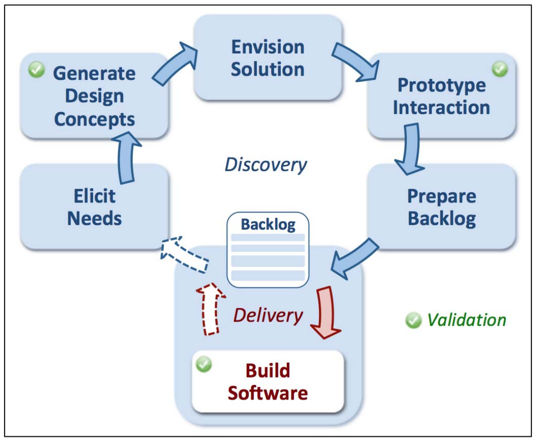


Fig. 2: Dual-track agile software development

The process involves two tracks: *Discovery*, aiming at discovering what to build, and *Delivery*, aiming at delivering working software. The two tracks run continuously and in parallel. For instance, at a given time, functionality *f1* could be implemented under **Build Software**, while the requirements for functionality *f2* are being detailed under **Prepare Backlog**, a functionality *f3* is being prototyped under **Prototype Interaction**, and the concept for a functionality *f4* is being designed under **Generate Conceptual Design**. Also, as illustrated in Figure 2, *Delivery* is part of *Discovery*, as one has not fully discovered what to deliver until the working system is validated by users.

Despite the fact that each activity has a leader, the whole team might be involved in all activities to avoid software development waste [46], including knowledge loss due to knowledge silos and handoffs.

The process activities can be summarized as follows:
- **Elicit Needs** – *Led by Iris*. Similar to what was described in Section III-B, through research and analysis, the team gains an empirical understanding of the stakeholders' needs.
- **Generate Design Concepts** – *Led by Iris*. Similar to what was described in Section III-B, through ideation and experimentation with storyboards and design walkthroughs, the team generates alternative viable interaction design concepts that could satisfy the stakeholders' needs.
- **Envision Solution** – *Led by Remi*. Based on the interaction design concepts that resonated the most with stakeholders during design walkthroughs, the team creates or updates a long-term product vision, and a list of prioritized high-level requirements (e.g., epics). As needed, Remi demonstrates the competitiveness of the new product ideas, and a software developer demonstrates their technical feasibility.
- **Prototype Interaction** – *Led by Iris*. Considering the highest-priority epic(s), the team continues experiment-

ing by creating and refining a low-cost medium-fidelity prototype and conducting usability testing sessions until obtaining a fully validated prototype.
- **Prepare Backlog** – *Led by Remi*. By taking the validated prototype as a guide, the team breaks down a few high-priority epics into smaller user stories with acceptance criteria. Remi adds and prioritizes the user stories into the product backlog.
- **Build Software** – *Led by Software Developers*. Iris and Remi present the validated prototype and high-priority stories to the software developers. Each developer selects and implements one or more stories located at the top of the backlog. Iris and Remi are available for clarification questions, pairing with developers as needed. The team is experimenting with code. For verification and validation purposes, and to discuss the direction of the product, the implemented stories are reviewed by the team, potential users, and other stakeholders.

While the *Delivery* track remains fairly constant, the focus of the *Discovery* track varies over the life of the product. As the product matures, the focus shifts away from exploratory activities, including interviewing and conceptual design, towards prototyping and usability testing. The team also relies more on analytics from working software. However, when new opportunities or questions arise, the team might revert to exploratory activities.

The scenario above illustrates a tight and effective collaboration among Iris, Remi, and the developers. By working together, Iris and Remi are able to combine their respective strengths and overcome the shortcomings of their traditional approaches. Each user story presented to the developers supports an idea for a functionality that is feasible, competitive, and validated by users. Many broken or less-compelling ideas have been eliminated along the way. As a result, the team is capable of delivering innovative products satisfying the users needs by being useful, usable, and delightful.

Finally, let us note that the above presentation of dual-track agile is oversimplified and does not do justice to the messy reality of software product design and development. An attempt to more effectively convey the disorderliness of dual-track agile is presented in [8]. However, simplicity comes handy when it comes to teaching and learning. In the next section, we show how this simple process has been used to guide the design of a new course integrating interaction design into software development.

IV. Course Design

The *Software Requirements and Interaction Design (SRID)* course was first designed and taught in 2015. It builds on two different courses previously taught by the author: one in industry and one in academia. The course is now in its fourth offering. Its design has evolved over the semesters. This section presents the course design of the last offering.

A. Learning Objectives

The learning objectives are as follows:

- Innovate with a design thinking mindset.
- Design software systems addressing real problems, satisfying real stakeholders' needs, and providing superior user experiences.
- In the context of dual-track agile software development, apply an integrated approach to RE and IxD.
- Understand the continuous and concurrent nature of product discovery and delivery.
- Plan and conduct effective user research.
- Perform competitive analysis.
- Apply design-based ideation and validation, including storyboarding, prototyping, and usability testing.
- Communicate the intent and scope of a software system to all concerned stakeholders.
- Collect customers' information using data analytics.
- Define actionable systems' software requirements so they could be easily implemented.
- Master a requirements management tool.

B. Overall Structure

The course is spread over 15 weeks, with 100-minute class sessions twice a week. It is designed to require on average 12 weekly hours of student effort, including participation in class sessions, class preparation, individual assignments, and team project components. The course outline is summarized in Table I. Class topics and activities are aligned with the tasks of a semester-long *Innovation* project. Each task focuses on one or more activities of the dual-track agile process presented in Section III-C. The continuous and concurrent nature of dual-track agile is illustrated throughout the semester using various scenarios, and experienced by students in Task 5. In addition, a guest lecture and a field project are added to bring the real-world to the classroom and provide an outside perspective.

TABLE I: Course Outline

Weeks	Innovation Project Tasks, Class Topics, Activities	
1-2	Introduction to RE, IxD, Dual-track Agile; Activity: Defining innovation projects and teams	**Dual-track Agile**
3-5	**Task 1: Needs Elicitation**; Activity: Conducting interviews; Activity: Synthesizing research data; Activity: Modeling personas and domain; Lab: Requirements management tool	
6-7	**Task 2: Conceptual Design**; Guest lecture on interaction design in industry; Design Thinking; Activity: Ideation; Activity: Storyboarding; Design walkthrough	
8-9	**Task 3: Solution Envisioning**; Product vision; Lean Startup; Landing page, concept video, data analytics; Activity: Conducting use case workshop; Activity: Conducting competitive analysis	
10-11	**Task 4: Prototyping & Backlog Preparation**; Rapid Prototyping; Usability testing; Activity: Medium-fidelity prototyping and user stories Activity: Planguage; Product backlog	
12-15	**Task 5: Continuous Product Discovery and Delivery**; Continuous discovery and delivery; Style guide; Usability evaluation; Lab: Heuristic evaluation; Final Innovation project presentations; Field project presentations; Exam	

C. Delivery Format

The delivery format is based on a mixed approach that combines traditional and flipped-classroom delivery in the context of project-based learning. Project-based learning [12] organizes learning activities around a project of realistic complexity. While working in teams on project tasks, students experiment with and reason about the underlying concepts of a discipline. Potential benefits include active participation in the learning process, promotion of critical thinking, development of soft skills, and a taste of real-world projects.

Flipped classroom [11] consists of individual instruction (typically video lectures) outside the classroom, and interactive group learning activities inside the classroom under the guidance of a faculty mentor. Flipping a project-based course could amplify its benefits by freeing-up time inside the classroom to better prepare students for the demands of the course project. However, because video lectures are initially long to create, and later on difficult to maintain, they have the propensity of freezing the course content. Consequently they tend to be incompatible with fast-evolving course subjects, like integrating interaction design into agile software development.

To overcome this problem in *SRID*, video lectures are replaced with selected readings and other relevant materials. In addition, a mini-lecture of about 15 to 30 minutes is added at the beginning of each class. The goal of the mini-lecture is to assess the students' understanding of the concepts introduced in the preassigned readings, by engaging them via questions and answers, and to clarify, deepen, and complement those concepts. The remaining of the class session is dedicated to an in-class activity where students apply the concepts to a toy example. That way, the course retains the benefits of flipped classroom without the drawbacks of video lectures. When time permits, students are encouraged to stay after the in-class activity to work on their task project under the guidance of the instructor.

Each project task is designed based on the principle that effective teaching involves aligning the three major components of instruction: *learning objectives*, *instructional activities*, and *assessments* [47]. Learning objectives are shared with students at the beginning of each task. Assessments are done in-class and/or via written feedback at the end of each task. As a result, and as illustrated in Figure 3, students are exposed to concepts from five different perspectives: (1) readings and other relevant materials, (2) mini-lecture with discussion, (3) application to toy example during in-class activity, (4) application to project task, and (5) task assessment. Each perspective builds on the previous ones to reinforce the students' understanding of the concepts. This reinforcement, without repetition, coupled with teamwork and targeted feedback, aims at enhancing learning and preparing students for real-world projects.

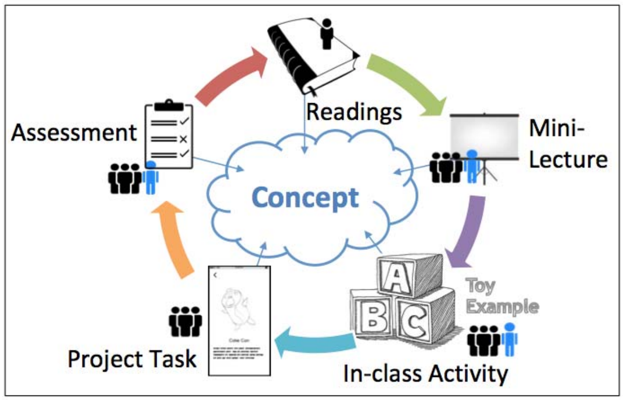


Fig. 3: Concept reinforcement, without repetition

D. Innovation Project

During the semester-long *Innovation* project, the students envision, prototype, and start implementing an innovative software system that could make a unique contribution to society. The system should address a real problem, satisfy real stakeholders' needs, and provide a superior user experience. Stakeholders should remain involved throughout the project.

1) Projects Definition and Team Formation: Students define and select their own projects. To prepare, each student is asked to contribute one social challenge that could potentially be addressed with a software system having a strong user focus, that students are capable of implementing, and for which they have access to potential users (ideally in professional settings).

During a class activity, starting with the most interesting challenges (e.g., about six challenges for a class of 30 students), and following the Charette Procedure brainstorming technique [48], students build on each other's ideas until they articulate a set of compelling product opportunities. Here is an example of opportunity identified by students during the class activity last semester: *"Modern society generates a lot of waste with a negative impact on our environment. We might be able to leverage software technology to decrease waste footprint through increased recycling."* Students artifacts presented in figures 5 and 6 relate to this opportunity.

A project team of four or five students is formed around each opportunity. Because motivation is a key factor of success in the course, students are asked to auto-form their team primarily based on project interest. Interpersonal relationship certainly plays a role as well for students who already know each other.

2) Project Tasks: The project tasks are designed based on the dual-track agile process presented in Section III-C. The project starts with the *Discovery* track.

- **Task 1, Needs Elicitation**, is organized around the **Elicit Needs** activity presented in Section III-C. Based on their unique opportunity, each student team researches the domain, identifies and interviews key stakeholders, and analyzes the data gathered during the interviews using affinity mapping, as illustrated in Figure 4, to empirically identify their own stakeholders' needs.
- **Task 2, Conceptual Design**, follows the **Generate Design Concepts** activity presented in Section III-C. Each student team applies design thinking to generate and validate alternative interaction design concepts that could

Fig. 4: *SRID* students performing affinity mapping

satisfy their stakeholders' unique needs. During ideation sessions, students are encouraged to think laterally and outside the box to come-up with a set of creative and original concepts. Each student individually illustrates a unique concept using a storyboard, as exemplified in Figure 5 with an original gamification concept to increase recycling. The students go back to their stakeholders in the context of design walkthrough sessions.

- **Task 3, Solution Envisioning**, follows the **Envision Solution** activity presented in Section III-C. Based on the interaction design concepts that resonated the most with stakeholders during design walkthroughs, each student team defines a product vision and a set of epics, among which they identify a Minimum Viable Product (MVP [49]) to be implemented first. In addition, each team conducts a competitive analysis to articulate how their product differentiates itself from the competition. Finally, each team starts collecting information about prospective customers using data analytics on a landing page showcasing their product vision and including a concept video.
- **Task 4, Prototyping and Backlog Preparation**, follows the **Prototype Interaction** and **Prepare Backlog** activities presented in Section III-C. Each team creates a wireframe-based click-through prototype for their MVP. The left-hand side of Figure 6 shows a wireframe of the prototype produced by the recycling team. The students go back to their stakeholders in the context of usability testing sessions until obtaining a validated prototype. Finally, taking the validated prototype as a guide, the team breaks down the corresponding epics into smaller user stories with acceptance criteria, prioritizes the stories, and adds them to the product backlog.
- **Task 5, Continuous Product Discovery and Delivery**, allows students to experience the continuous and concurrent nature of dual-track agile. They start the *Delivery* track by implementing a working web application for their team's MVP, while continuing the *Discovery* track to discover what to build next and update the product backlog accordingly. In order to deliver a first valuable

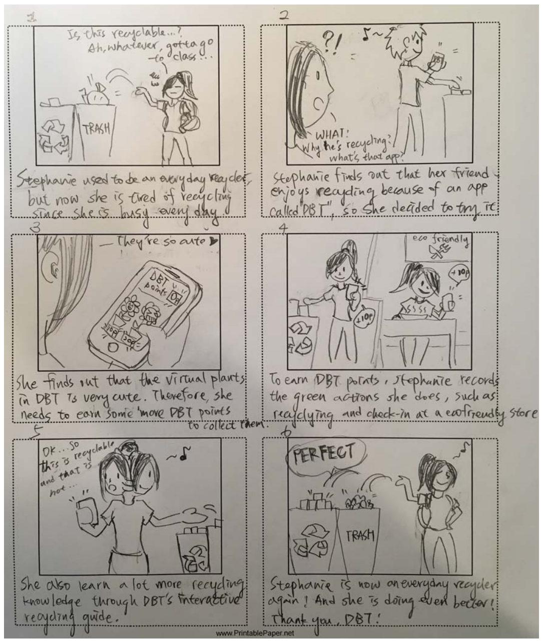
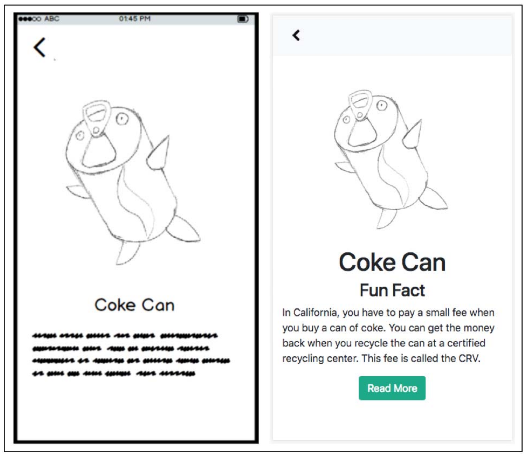

1
Is this recyclable...?
Ah, whatever, gotta go to class...
TRASH
Stephanie used to be an everyday recycler, but now she is tired of recycling since she is busy every day.
2
?!
WHAT! Why he's recycling? what's that app?
Stephanie finds out that her friend enjoys recycling because of an app called "DBT", so she decided to try it.
3
They're so cute
DBT points
She finds out that the virtual plants in DBT is very cute. Therefore, she needs to earn some more DBT points to collect them.
4
eco friendly
+10P
To earn DBT points, Stephanie records the green actions she does, such as recyclying and check-in at a ecofriendly store
5
OK... So this is recyclable and that is not...
She also learn a lot more recycling knowledge through DBT's interactive recycling guide.
6
PERFECT
TRASH
Stephanie is now an everyday recycler again! And she is doing even better! Thank you, DBT!
www.PrintablePaper.net

Fig. 5: Storyboard illustrating a recycling gamification concept

product increment, each student selects and implements a few stories located at the top of the backlog. The application should include a fully functional, useful, usable, and delightful front-end implemented with HTML/CSS/JS. However, because of time constraints, the back-end might include hard-coded functionality and data. The right-hand side of Figure 6 shows a screen of the mobile application developed by the recycling team. Each team then checks that the application of another team complies with recognized usability principles via heuristic evaluation [50]. Students contact their stakeholders for the last time, sending the link to their application and asking for feedback. Concurrently, they revisit some



Coke Can

Fun Fact

In California, you have to pay a small fee when you buy a can of coke. You can get the money back when you recycle the can at a certified recycling center. This fee is called the CRV.

Read More

Fig. 6: From prototype to working recycling application

activities performed in previous tasks in order to continue discovering what to build next and to contribute new user stories to the product backlog. Finally, each team pitches and demonstrates its product during a final presentation to the class.

One notable aspect of the project is the high level of stakeholder involvement. Each interaction with stakeholders is an opportunity to pivot the product direction to better satisfy their needs. Also notable, is the fact that each team goes beyond creating an interaction design addressing their opportunity by preparing a backlog for implementation. Furthermore, the team delivers the first valuable increment of a working application while getting a taste of the continuous and concurrent nature of product discovery and delivery.

E. Field Project

The field project aims at bringing the real-world to the classroom by reporting on the state of RE and IxD at selected Silicon Valley companies. To that end, each student team is asked to interview two practitioners working at a same company and on the same product: one practitioner in charge of requirements (e.g., Requirements Engineer, Product Manager, Product Owner, Analyst) and one practitioner in charge of user experience (e.g., Interaction Designer, Product Designer, User Experience Designer). Following the interviews, student teams share their findings with the class, including a comparison of what they learn in class with what they observed in the field. Companies that have been investigated during past semesters include, among others: Amazon, Facebook, Google, Linkedin, Microsoft, Netflix, Quantcast, and SAP. These investigations highlight a wide diversity of software development approaches and maturity. Only a minority of these companies so far have specifically mentioned dual-track agile as their core software development process. Overall, the field project provides students with a first-hand exposure to practitioners dealing with RE and IxD, and their perspectives on these domains. It is an opportunity to improve the students' interview and communication skills while enlarging their professional network.

V. Challenges

Overall, the course is well received by students, as illustrated by the following quotes taken from the online course evaluation: *"Excellent course with great teaching style combining industry practice with classroom theory."*; *"The course is deeply connected with the project, so we can 'learn by doing'. Really enjoy it, thank you!"*; *"Great Course, adds a lot of value to me as a software engineer."*; and *"Great course that demonstrates the importance of working hand in hand with users to understand their needs. I would recommend this course to everyone!"* However, the course also comes with its challenges. This section covers past and current challenges, together with attempts to overcome them.

A. Project Selection

In the context of project-based learning, the success of the course relies on finding motivating projects that effectively support the course learning objectives. To reach the *SRID* course objectives, students must work on a software system that has a strong user focus, that they are capable of implementing, and for which they have access to potential users (ideally in professional settings).

To make sure that these requirements were satisfied, during the first two offerings of the course, projects were imposed on students. The theme was *emergency response*, and each team had to define a software system for a different type of response team (e.g., fire, police, dispatch, emergency medical services). Contact information of stakeholders to interview was provided. While most students enjoyed working on these projects, some complained about the lack of flexibility.

Therefore, in subsequent and current semesters, students are encouraged to define their own projects (see Section IV-D1). The only recommendation is to identify opportunities related to problems with important societal impact, with the hope of raising students' social and civil awareness. Students are responsible for identifying and contacting stakeholders to interview. This approach generates a higher level of student engagement and motivation. Working on one's own idea is exciting. Even though some students are nervous about finding and contacting stakeholders on their own (with only the help of a provided recruitment email template), all teams have managed to do so fairly successfully.

Finally, most teams have proposed project ideas with a goal of making a unique contribution to society. The domains of these projects have included education, food waste, homelessness, medicine, natural disaster relief, recycling, traffic congestion, and the elderly.

B. Project Documentation

A course like *SRID* involves many different interrelated artifacts, including research artifacts, interaction design artifacts, requirements artifacts, and software system artifacts. For instructors who need to explain what needs to be produced, and later evaluate these artifacts, the overhead could be significant. The same goes for students, who need to understand what to produce, and create and organize these artifacts effectively so they can be shared across the team and with instructors.

In *SRID*, the problem was eventually solved by introducing a requirements management tool called *Jama* [51]. The tool serves as a hub for all project artifacts. Its content structure has been fully customized to perfectly match the structure of the course project, helping students understand how various elements fit together. For each project artifact, the tool provides guidance, examples, and potentially templates to jump-start its creation. Project artifacts can be accessed from different perspectives to achieve different goals, including creating, evolving, versioning, prioritizing, filtering, and sorting projects artifacts. Discussion threads could be created around individual artifacts for enhanced collaboration.

Overall, the tool reduces the overhead of structuring the information, at both project and artifact levels, as well as the overhead of understanding what content to produce. As a result, students can focus on content creation instead.

C. Effective Feedback

In the context of project-based learning, the quality of the work done on one artifact, or during one task, might impact subsequent artifacts or tasks. Consequently, feedback must be timely and targeted in order to be effective and guide further learning. Failing to so do might compromise students' ability to succeed.

Here as well, the requirements management tool has helped. The tool provides instructors with an improved visibility of the students' work throughout the project. At any time, the instructor can assess students' progress by looking at deliverables completed or still under construction, including revision history and attribution to get a sense of who is doing the work. Instructors can have offline discussions with students related to any specific section of a deliverable.

At the end of a task, each team packages the task's deliverables using the tool's review mechanism. This allows instructors to formally evaluate the entire work done during the task while easily providing targeted feedback. The process in itself is a learning experience for students, as it mimics real-world industry reviews.

Overall, the tool improves the instructor's ability to effectively collaborate with students outside the classroom, for both mentoring and evaluation purposes. It also supports the evaluation of individual contributions in the context of team projects. A summary of how the tool is leveraged in the context of *SRID* is available on the Jama website [52].

As a result of offline feedback via the tool along with mentoring in the classroom, feedback has always scored high in the online course evaluation, even with a class of 33 students. Still, providing effective feedback becomes challenging as class size grows. Evaluating creative and novel work that is tackling open-ended problems is time consuming. Furthermore, when left on their own devices, students tend to revert to old habits, happily preconceiving problems and solutions, building a product for the first solution that comes to mind, and avoiding talking to users until the first version of the product is fully implemented. Preventing that from happening requires constant attention and guidance. With one instructor and one teaching assistant, a class size of up to about 20 student is ideal. Above that number, the quality of teaching and learning might be compromised.

D. Continuous Product Discovery and Delivery

Many RE and IxD practices introduced in the course are challenging for students, especially when these practices involve collaborating with stakeholders. Learning to apply RE and IxD practices effectively is time-consuming. As a result, each practice might be applied only once during the semester, with little or no time left for implementation. Consequently, students might be under the impression that the project follows the waterfall approach.

In *SRID* this challenge was eventually addressed by framing the RE and IxD practices within dual-track agile. The introduction of each task of the *Innovation* project (see Section IV-D2) is an opportunity to remind students about the continuous and concurrent nature of the approach. In **Task 5, Continuous Product Discovery and Delivery**, students deliver the first valuable increment of a working software. To minimize ramp-up, they leverage the technology and development practices learned during a previous course of the program (Foundations of Software Engineering [9], which focuses on 'traditional' agile development and continuous delivery). Concurrently, they briefly reapply some RE and IxD practices to continue discovering what to build.

This strategy provides students with a taste of dual-track agile and helps them understand the continuous and concurrent nature of product discovery and delivery, while still allocating enough time to properly learn the challenging RE and IxD practices. Ideally, however, a course like *SRID* should be followed by another course (e.g., practicum, capstone) where students can apply their newly acquired knowledge to drive the continuous and concurrent discovery and delivery of a series of valuable product increments throughout the semester, and hence fully experience dual-track agile.

VI. Conclusion

This paper presents a novel course integrating interaction design into software development. It documents the author's experience designing and teaching the course over the past four years. The paper makes the following contributions:

- It documents the disconnect between the SE community's focus on *how to build* and the HCI community's focus on *what to build*, in both industry and academia.
- It analyzes the strengths and weaknesses of 'traditional' agile software development.
- It describes how user experience teams traditionally go about supporting software development, together with the strengths and weaknesses of their approach. Among the strengths, it emphasizes divergent thinking and coevolution, and their fundamental role in the design process.
- It describes dual-track agile using a novel process model, and shows how dual-track agile combines the respective strengths of traditional agile and user experience approaches while overcoming their shortcomings.
- It reveals the design of a course that uniquely integrates interaction design and requirements engineering in the context of dual-track agile, and teaches students to design and implement software systems addressing real problems, satisfying real stakeholders' needs, and offering superior user experiences.
- It presents a rarely documented delivery format based on a mixed approach combining flipped-classroom and traditional delivery in the context of project-based learning.
- It shares the challenges involved in teaching the course.

These contributions are important for software engineering educators in higher-education settings because, despite the emergence of dual-track agile in industry, it appears that no publication so far has reported the existence of such a course in academia. We hope that the paper will convince, inspire, and enable others to teach similar courses, hence bringing user experience to the forefront of agile software development.

References

[1] A. Seffah and E. Metzker, "The obstacles and myths of usability and software engineering," *Communications of the ACM*, vol. 47, no. 12, pp. 71–76, Dec. 2004.

[2] R. Hartson and P. Pyla, *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

[3] A. Cooper *et al.*, *The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity*. Sams Indianapolis, 2004.

[4] R. Hinman, *The mobile frontier: a guide for designing mobile experiences*. Rosenfeld Media, 2012.

[5] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review, systematic review paper," *Information and Software Technology*, vol. 61, no. C, pp. 163–181, May 2015.

[6] K. Albrecht. (2015) Dual track agile: Focusing on customer value. [Online]. Available: https://medium.com/kevin-on-code/dual-track-agile-focusing-on-customer-value-a2e39312585b

[7] P. Ralph, "Improving coverage of design in information systems education," in *Proceedings of the 2012 International Conference on Information Systems*, 2012.

[8] T. Sedano, P. Ralph, and C. Péraire, "The product backlog," *Accepted at the 41th International Conference on Software Engineering, ICSE'19*, 2019.

[9] H. Erdogmus and C. Péraire, "Flipping a graduate-level software engineering foundations course," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, ser. ICSE-SEET '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 23–32.

[10] H. Erdogmus, S. Gadgil, and C. Péraire, "Introducing low-stakes just-in-time assessments to a flipped software engineering course," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[11] M. J. Lage, G. J. Platt, and M. Treglia, "Inverting the classroom: A gateway to creating an inclusive learning environment," *The Journal of Economic Education*, vol. 31, no. 1, pp. 30–43, 2000.

[12] H. A. Hadim and S. K. Esche, "Enhancing the engineering curriculum through project-based learning," in *Frontiers in education, 2002. FIE 2002. 32nd Annual*, vol. 2. IEEE, 2002, pp. F3F–F3F.

[13] K. Schwaber and J. Sutherland. (2015) Scrum guide. [Online]. Available: http://www.scrumguides.org/

[14] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[15] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[16] C. Larman and B. Vodde, *Large-scale scrum: More with LeSS*. Addison-Wesley Professional, 2016.

[17] D. Leffingwell, *Scaling software agility: best practices for large enterprises, Chapter 17: Lean Requirements at Scale: Vision, Roadmap, and Just-in-Tine Elaboration*. Pearson Education, 2007.

[18] K. Wiegers and J. Beatty, *Software requirements, Third Edition*. Pearson Education, 2013.

[19] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About Face : The Essentials of Interaction Design*. Somerset: John Wiley amp; Sons, Incorporated, 2014.

[20] H. Plattner, "An introduction to design thinking, process guide," *Institute of design at Stanford*. [Online]. Available: https://dschool-old.stanford.edu/sandbox/groups/designresources/wiki/36873/attachments/74b3d/ModeGuideBOOTCAMP2010L.pdf

[21] D. Council, "The design process: What is the double diamond?" [Online]. Available: https://www.designcouncil.org.uk/news-opinion/design-process-what-double-diamond

[22] B. Martin and B. Hanington, *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport Publishers, 2012.

[23] A. Seffah, J. Gulliksen, and M. C. Desmarais, *Human-centered software engineering-integrating usability in the software development lifecycle*. Springer Science & Business Media, 2005, vol. 8.

[24] D. Salah, R. F. Paige, and P. Cairns, "A systematic literature review for agile development processes and user centred design integration," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14. New York, NY, USA: ACM, 2014, pp. 5:1–5:10.

[25] C. Salvador, A. Nakasone, and J. A. Pow-Sang, "A systematic review of usability techniques in agile methodologies," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, ser. EATIS '14. New York, NY, USA: ACM, 2014, pp. 17:1–17:6.

[26] J. De Litchenberg. (2017) Dual-track agile: Why messy leads to innovation. [Online]. Available: https://www.mindtheproduct.com/2017/04/dual-track-agile-messy-leads-innovation/

[27] J. Patton. (2017) Dual track development is not duel track. [Online]. Available: https://jpattonassociates.com/dual-track-development/

[28] L. Miller, "Case study of customer input for a successful product," in *Proceedings of Agile 2005*. IEEE, 2005, pp. 225–234.

[29] D. Sy, "Adapting usability investigations for agile user-centered design," *J. Usability Studies*, vol. 2, no. 3, pp. 112–132, May 2007.

[30] R. J. LeBlanc, A. Sobel, J. L. Diaz-Herrera, T. B. Hilburn *et al.*, *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. IEEE Computer Society, 2006.

[31] H. Topi, J. S. Valacich, R. T. Wright, K. Kaiser, J. F. Nunamaker Jr, J. C. Sipior, and G.-J. de Vreede, "Is 2010: Curriculum guidelines for undergraduate degree programs in information systems," *Communications of the Association for Information Systems*, vol. 26, no. 1, p. 18, 2010.

[32] K. Dorst and N. Cross, "Creativity in the design process: co-evolution of problem–solution," *Design studies*, vol. 22, no. 5, pp. 425–437, 2001.

[33] P. Ralph, "Re-imagining a course in software project management," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ACM, 2018, pp. 116–125.

[34] B. Buxton. (2007) Sketching and experience design. Stanford University Human-Computer Interaction Seminar. [Online]. Available: https://www.youtube.com/watch?v=xx1WveKV7aE

[35] Y. D. Pham, D. Fucci, and W. Maalej, "A first implementation of a design thinking workshop during a mobile app development course project," in *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*. ACM, 2018, pp. 56–63.

[36] N. M. C. Valentim, W. Silva, and T. Conte, "The students' perspectives on applying design thinking for the design of mobile applications," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 2017, pp. 77–86.

[37] M. Palacin-Silva, J. Khakurel, A. Happonen, T. Hynninen, and J. Porras, "Infusing design thinking into a software engineering capstone course," in *Software Engineering Education and Training (CSEE&T), 2017 IEEE 30th Conference on*. IEEE, 2017, pp. 212–221.

[38] D. Ogle. (2009) Contextual inquiry. [Online]. Available: https://wiki.fluidproject.org/display/fluid/Contextual+Inquiry

[39] (2012) The affinity diagram tool. Six Sigma Daily. [Online]. Available: http://www.sixsigmadaily.com/the-affinity-diagram-tool/

[40] B. Buxton, *Sketching user experiences: getting the design right and the right design*. Morgan Kaufmann, 2010.

[41] S. Greenberg, S. Carpendale, N. Marquardt, and B. Buxton, "The narrative storyboard: telling a story about use and context over time," *interactions*, vol. 19, no. 1, pp. 64–69, 2012.

[42] L. Caballero, A. M. Moreno, and A. Seffah, "Persona as a tool to involving human in agile methods: contributions from hci and marketing," in *International Conference on Human-Centred Software Engineering*. Springer, 2014, pp. 283–290.

[43] R. Bias, "Interface-walkthroughs: efficient collaborative testing," *IEEE Software*, vol. 8, no. 5, pp. 94–95, 1991.

[44] T. Sedano, P. Ralph, and C. Péraire, "Sustainable software development through overlapping pair rotation," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement International Conference on Software Engineering*, ser. ESEM, 2016.

[45] ——, "Practice and perception of team code ownership," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE. ACM, 2016.

[46] ——, "Software development waste," in *Proceedings of the 2017 International Conference on Software Engineering*, ser. ICSE '17. IEEE, 2017.

[47] Teaching principles. Eberly Center for Teaching Excellence and Educational Innovation, Carnegie Mellon University. [Online]. Available: https://www.cmu.edu/teaching/principles/teaching.html

[48] R. Elmansy. Brainstorming multiple ideas using charette procedure. [Online]. Available: https://www.designorate.com/brainstorming-using-charette-procedure/

[49] E. Ries, *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

[50] J. Nielsen and R. Molich, "Heuristic evaluation of user interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1990, pp. 249–256.

[51] Jama software. [Online]. Available: https://www.jamasoftware.com

[52] M. Wigen Kernan. Foundations for modern requirements management. Jama Software. [Online]. Available: https://www.jamasoftware.com/blog/modern-requirements-management-at-cmu/