Chapter 1
Integrating User-Centred Design in Agile Development

Gilbert Cockton, Marta Lárusdóttir, Peggy Gregory, and Åsa Cajander

Abstract Integrating user-centered design (UCD) into software development methodologies has always been a challenge. In the first two decades of UCD, structured methodologies provided a process where UCD methods could be clearly integrated, at least in principle, and thus integration challenges were not primarily due to process structures. This changed with the spread of agile software development approaches, which differ substantially from structured development. While there has been progress in combining UCD and agile approaches, many problems remain. In response to this, at NordiCHI 2014, a workshop on *Integrating User Centred Design in Agile Development* brought together researchers and practitioners at the leading edge of combining these two potentially complementary approaches to software development. The chapters in this book update and extend the position papers that were presented and discussed at the workshop. Six authors developed their position papers into chapters for this book. Five additional chapters introduce the book, report on the initial workshop, and provide three additional studies. The case studies in this book cover a very wide range of organizational sizes (from 8 to 135,000) in over 15 countries, operating in consumer, Small and Medium Enterprises (SME), large Business to Business (B2B), public sector and non-profit markets across a range of around 20 application domains. Some chapters synthesise several studies that were conducted over a number of years. This introduction presents the context for the workshop, identifies common themes with regard to cultures, teams and tasks in agile UCD development, and discusses future trends and a research agenda for adapting UCD to agile development contexts.

G. Cockton (✉)
School of Design, Northumbria University, Newcastle upon Tyne NE1 8ST, UK
e-mail: gilbert.cockton@northumbria.ac.uk

M. Lárusdóttir
School of Computer Science, Reykjavik University, Reykjavik, Iceland

P. Gregory
School of Physical Sciences and Computing, University of Central Lancashire, Preston, UK

Å. Cajander
Department of Information Technology, Uppsala University, Uppsala, Sweden

G. Cockton et al. (eds.), *Integrating User-Centred Design in Agile Development*,
Human–Computer Interaction Series, DOI 10.1007/978-3-319-32165-3_1

Keywords User-centered design • User experience • Participatory design • Design thinking • Agile software development • Scrum • Future trends

1.1 User-Centred Software Development

Integration of User-Centred Design (UCD) within agile systems development processes (Agile) can be problematic. Although some persistent difficulties predate Agile, its evolving contexts continue to constrain possibilities for integrating UCD activities. Even so, there has been much progress over the last decade. The number of papers considered in systematic literature reviews about this topic has risen from 35 to 83 across 2010, 2011, 2014 [1] and 2015 (See Chap. 8 for references). These papers identify the conditions under which Agile and UCD can work together. Agile claims to consider the users' perspective, but this is not always the case. There are still opportunities for innovation, research and discussion.

The integration of UCD with software development processes such as 1980s structured methods was challenging, but for different reasons to those currently associated with Agile. As software engineering (SE) developed in the 1970s, it focused on software artefacts, much as older engineering practices focused on their artefacts too, such as bridges, dams, manufacturing lines and power generators. Waterfall processes dominated, which were based on a fixed sequence of development phases, from problem analysis and requirements specification, through design, implementation and verification, to software installation and operation [2].

UCD was initially developed and advocated to improve software usability. As hardware costs plummeted in the 1980s, more people had access to computer equipment, but did not have access to the extensive training previously given to specialist computer operators. After effective usability methods were developed, in the 1990s UCD's scope broadened to quality in use. Satisfaction, effectiveness and contextual fit were added to speed, and ease of use and learning as foci for UCD work. In the 2000s, UCD's quality in use foci further extended to include positive user experience (UX). With each extension of UCD's scope, the aim has been to improve quality in use, both by reducing costs and risks and also by better addressing users' experiences, needs and wants. Today, UCD's benefits can convincingly deliver on a wide range of values, including effectiveness, efficiency, satisfaction, wellbeing at work, brand loyalty, health and safety, employee retention, respect for human dignity, and competitiveness. Given this, in theory UCD should be attractive to software development.

When attempts were first made to introduce UCD into software development in the 1980s, the main challenge was gaining acceptance for UCD practices across all development phases. UCD aligned well with waterfall processes, but the strength of the case for UCD was often not enough to extend UCD activities beyond evaluation. UCD activities such as contextual research fitted into the problem analysis phase, and user testing fitted into the verification phase. However, UCD required development phases to be repeatedly iterated [2] until the software under development satisfied user needs for both functionality and usability. This

required time to be set aside for regular communication and decision-making with stakeholders. Not all software methodologies were capable of being made iterative and consultative [2].

Agile developed as a reaction against the constraints of waterfall software development methodologies. Some of its key characteristics were compatible with UCD: evolving requirements iteratively across a series of iterations, incremental development to allow early delivery of working software, and introducing close collaboration with customer representatives. Other characteristics were less favourable to UCD, such as reduced opportunities for user testing and less upfront planning before software implementation. These adverse factors can almost exclude UCD from the most popular agile practices [3, 4], or have UCD responsibilities transferred to business analyst roles. However, there are several different agile methodologies [3, 4], so it is important to avoid generalizations that do not apply to all of them. Furthermore, development practices descriptions are (knowingly) unavoidably underspecified, so all methodologies must be contextualized and completed in use. Thus neither positives nor negatives that are apparent in published descriptions of a methodology will inevitably emerge in practice. Project teams have to work at achieving potential positives, and they can also work to avoid potential negatives. What actually happens during software development is more important than what a methodology advocates or what detractors say will happen. It is thus important to go beyond published accounts to understand how agile development is managed, and how this impacts the ability to integrate UCD activities. Nevertheless, published methods do set the tone and expectations for practice. The limitations that we see in published methods are often not overcome in practice.

1.1.1 Position Papers and Chapters

This book presents case studies and forward-looking analyses of the potential for improved integration of UCD activities within Agile. Most of the chapters began as position papers for a NordiCHI 2014 workshop [5]. The call for participation for this workshop invited position papers on:

- Case studies and work in progress related to UX and Agile.
- Success stories and best practices from integrating UCD and Agile.
- Challenges from working with UCD in Agile
- Integration of UCD and Agile in different domains such as games and healthcare.
- Values and perspectives underpinning UCD and Agile in theory.
- Theories and methods relevant to research on Agile and UCD.
- Discussion of future trends for UCD and Agile research.

There is an overview of the position papers in Chap. 8, which summarizes the results of the NordiCHI 2014 workshop. Two position papers presented at the workshop were not developed into chapters for this book: one by Lindell [6] and one by Law and Lárusdóttir [7]. In [6], the design process of a music creativity app was described. Workshop discussion of this paper identified issues with breaking

down the 'big idea' for the app into manageable and meaningful small units for development in a single agile 'sprint' (iteration). Such issues, which compound the impact of limited upfront design of architectures, are a known problem with the most popular agile methodologies [4]. At first in [6], design and implementation work focused on technology, but this left many design questions unanswered. Lindell argued for a quality-driven open-ended artisan approach to software development, with careful attention to the experiential qualities of the design.

The position paper by Law and Lárusdóttir [7] presented and discussed an analysis of whether agile software development, exemplified by Scrum, is more compatible with UX work than lean development, exemplified by Kanban. The comparison between Scrum and Kanban, using a rather lean and agile analysis process, led the authors to make a preliminary conclusion that Kanban fits UX better, given its greater flexibility. Subsequently the authors conducted a more comprehensive study on the same subject [8], but the results of empirical studies were ambivalent. Neither Kanban nor Scrum support UX effectively.

There are summaries for each chapter in the Book Overview before this introductory chapter. Two case study chapters report on first time use of Scrum in a UCD context, and contain useful tutorial material for readers who are new to Scrum: Chap. 4 (Sect. 4.2.2) and Chap. 5 (Sect. 5.3.1). For readers new to UX, Chap. 7 (Sect. 7.2) provides similar tutorial material in relation to its study of agile teams' ability to predict users' evaluations of UX and need satisfaction.

The overall objective of the workshop was to provide a venue for researchers and practitioners, from within and outside of Human-Computer Interaction (HCI), to begin to shape the future of Agile and UCD research. The workshop had two goals:

1. Identifying future trends for research on Agile and UCD
2. Identifying challenges and success stories when working with UCD and Agile.

Most accepted position papers responded to the second goal. Six workshop position papers have been revised, extended and strengthened to provide detailed case studies for this book (Chaps. 2, 3, 4, 5, 6 and 9). For example, Chap. 3 extends its initial position paper with the outcomes of six additional iterations of UCD training materials for use in Agile contexts (doubling the underpinning evidence since the workshop). Chapter 7 is an additional study by a workshop attendee and her colleagues.

The extensive study of diverse Agile UCD practices is a distinctive strength of this book. The workshop summary (Chap. 8) benefits from this breadth, as do other forward-looking chapters (Chaps. 9, 10 and 11). Chapter 9 updates a future oriented position paper on a new framework for Agile UCD. Chapters 10 and 11 from three of the editors and their colleagues add two analyses that reflect critically on the futures of both UCD and Agile (from the positions of Management Fashion and Creative Design respectively). Chapters 9 and 10 draw extensively on several existing case studies by their authors. Chapter 11 draws on creative design research literature to challenge the position that a simple combination of UCD and Agile is sufficient for software innovation. All chapters are informed by the workshop, which at least one author of each chapter attended.

The case studies across the book's chapters span a very wide range of organizations. The smallest has eight employees, with most in the range of 100–500 or 500–2000. The largest range from 10,000–20,000 to 135,000 worldwide. The case study companies have staff in over 15 countries, operating in consumer, SME, large B2B, public sector and non-profit markets across a range of over 20 application domains, including:

> home and SME finances, home technologies and appliances (including smart homes), automotive technologies, mobile technologies (especially apps), telecommunications, energy and power systems, manufacturing systems, enterprise software (including customer process monitoring and licensing), banking, e-commerce, healthcare, music technology, nursery schools, higher education, local community systems, web portals for information aggregation, web infrastructure software for sign-on and app launching, data integration/visualization, embedded systems, cross platform systems, and software testing tools.

As noted in Chap. 8 (workshop summary), the breadth of application areas here indicates that Agile is now mainstream. Many case study applications were business critical and a few were safety critical. Some participating companies were bespoke system developers for external customers, whereas others developed in-house and consumer systems. Several involved hardware as well as software development. Around 450 respondents in total participated in the case studies reported and analysed in this book. All case studies (Chaps. 2, 3, 4, 5, 6 and 7) involve Scrum users, with Chaps. 4, 5 and 6 reporting projects using Scrum for the first time.

Chapter 8's themes, which emerged from the workshop presentations of position papers, are thus well grounded in current UCD practices within Agile. These themes are extended and deepened by the additional breadth and detail of the book chapters. However, the value of breadth here does not just lie in convergence on repeated themes, but also in the distinctive practices that were unique to projects and/or organisations. These distinctive practices resist simple generalisations and demand an open contextual approach to understanding relationships between UCD and Agile.

The breath across some aspects of the position papers did however allow some common themes to be identified at the workshop. These have been reinforced and extended by this book's chapters. They relate to both current and past agile UCD practice, and also to future possibilities for research and practice. The next section presents themes for current and past agile UCD practice. Sect. 1.3 discusses future trends.

1.2 Main Common Themes for Current and Past Agile UCD Practice from the Workshop and Chapters

The chapters in this book update and extend the position papers from our NordiCHI 2014 workshop. The chapters let us revisit, strengthen and extend observations made during the presentations of position papers. As reported in Chap. 8 (workshop summary), position papers reported on best practice, ongoing challenges, and future

opportunities. These provided items for an affinity diagram that was simplified after the workshop into eight themes across two groups: "challenges and obstacles", which have been experienced and will persist into the present; and "interesting points" which can be in the past (e.g., as best practice to adopt) or future (e.g., as opportunities). Themes can thus orient towards either the past or future.

Chapter 8 reviews the identified themes in detail. In this section, we use a higher level overview based on Chap. 9 (novel framework for Agile UCD), which recommends a shift of focus from *team roles* to *tasks* when improving Agile UCD. Chapter 8's themes from the workshop are thus further consolidated into observations about *teams* and *tasks*. These are alternative but overlapping lenses on software development, one focusing on how development teams are organized and managed, and the other focusing on how development *work* is organized and managed. The two are highly interdependent and in reality they are largely inseparable. However, some issues and opportunities are best approached from a team perspective and others are best approached from a work perspective. There are some loose couplings here, in that some aspects of process structure can be common to several team structures, and vice versa.

Table 1.1 associates example workshop themes and chapter topics with *teams* and *tasks*. Three workshop themes are not included: "miscellaneous"; *Culture*

Table 1.1 Team and task topics across chapters

Chapter	Team topics	Task topics in chapter
2	Roles	Channels and tools usage
		Filtering and interpretation
3	Training	Hands-on use
4	Team	Artefacts (within tasks)
		Activities
5	Customer Committee	Inception
	Scrum Islands	Sprint n.0
		One-week sprint
		(IN)Sprint Review
		Project retrospective
6	User involvement	Document use
		Synchronisation
7	PO, UX specialists and developers	UX assessment
8	*Post workshop themes:*	Artefacts and tools
	People and Roles	Methods and practices
	Teams and Communication	Time and synchronisation
9	People	Process
		Tasks
		Tools
10	Insiders and Outsiders	Activities
11	Generosity	Talkback
		Problem-solution co-evolution

("challenges and obstacles"); and *Research and Problems* ("interesting points"). The latter two are respectively covered before and after team and task factors. This section thus gathers workshop themes under the headings of:

1. Culture
2. Teams
3. Tasks
4. Research

The fourth theme was not a major focus at the workshop, but it is addressed in detail across the chapters in this book, as summarised in Sect. 1.2.4 below.

1.2.1 Culture

In 2002, Elden Nelson interviewed Kent Beck ("the father of XP") and Alan Cooper (inventor of Goal-Directed Design and Personas) [9]. The aim was to find the common ground and the points of difference between the agile approach of XP and the UCD approach of Goal-Directed Design. Relatively little common ground emerged during the interview, with points of major differences throughout. Ambler's reading of this interview was that "our thought leaders may be a bit too extreme" [10]. It is a clear example of a common tendency to exaggerate and entrench the culture clash between Agile and UCD. This clash can make integration of Agile and UCD appear to be impossible, but it is clear from the case studies in this book that this is not so, although no-one is claiming that a perfect marriage has yet been achieved. It has proved impossible for many projects and organisations to fully integrate UCD into agile practices. Chapter 8 notes the tensions between designer and developer cultures that surfaced at the workshop. Chapter 7 (study of agile team roles predictions of users' UX evaluations) captures some of these tensions: UCD does not trust developers to understand users, but developers do not trust users to know what they want.

Underpinning values and perspectives were a focus in the workshop call, and can continue to obstruct worthwhile innovations, both at the level of specific projects and in the wider professional communities where developers and designers form their identities and allegiances. While many synergies between Agile and UCD are being successfully exploited, remaining areas of conflict need to be identified, understood and (where possible) effectively addressed. This has to be related to the wider context of digital products and services, and related to other professional cultures such as Participative Design and Design Thinking (Chap. 6).

Cultures can only manifest themselves through people's practices in specific places. Our behaviours, built environments and material artefacts manifest meanings, and thus values (Chap. 10 focuses on why Agile is valued). Manifestations of culture in turn shape people's expectations and behaviours. Organisations that

wholly base development work on Agile or UCD can readily appear to have practices that could never be compatibly combined, as they take different positions on:

- what constitutes a valid problem or issue and what can be ignored
- what resources are provided and how they are allocated
- adequacy and excellence in design work

Values across the chapters that can underpin incompatible practices, create synergies, or even both, are now reviewed.

1.2.1.1 Agile and UCD Values

Book chapters present value systems that are disjoint, but not automatically incompatible. Agile values (in the sense of things that are valued) across chapters include:

- self-empowered independent autonomous teams of interacting individuals
- working software that satisfied customers accept for the value that it delivers
- customer collaboration, with acceptance of changing requirements
- flexible practices for rapid reaction to customer feedback
- velocity evidenced in early delivery of initial working versions, speed to market, and speed of communication within a project with on-time delivery
- visibility, awareness and accountability
- productivity: ability to focus without interruption
- a sense of achievement in relation to clear roles and short term goals (in contrast, developers can see usability as vague and fuzzy)
- ease of development with low waste, preference for informal lightweight methods
- being fashionable

These are values as communicated in the chapters. They are not necessarily all orthodox for all agile methodologies. For example, speed is not promoted as a value in the Agile manifesto [11]. What is valued there is as the highest priority is "early and continuous delivery of valuable software." Similarly, being fashionable is not mentioned in the manifesto either. In contrast, UCD values expressed across the chapters include:

- iterative processes and tools that support planning of comprehensive user-focused research and objective empirical evaluation, maintaining a user focus at all times
- human science values evidenced through well documented evidence and data analysis in user research and usage evaluation, removing 'games of chance' from design
- understanding the user before meaningful software development begins
- a coherent holistic picture of what will be developed
- superior expert knowledge of human-computer interaction (HCI)
- attention to detail: few usability problems should be left behind!
- satisfied users, due to hedonic and instrumental quality in use (UX values)

The above two lists of worthwhile practices and attributes from this book's chapters go well beyond the Agile Manifesto's [11] compact four pairs of more and less favoured phenomena of:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Two things are important to note here. Firstly, the manifesto begins: "we are uncovering better ways of developing software by doing it and helping others to do it". The manifesto is a contribution to a process, not a last word. However, this is not how it is always read, especially after 15 years. Secondly, there are no inherent value clashes here, since the manifesto states that "while there is value in the items on the right, we value the items on the left more" [11]. The manifesto itself thus does not outlaw the processes, tools, documentation, contracts or plans "on the right". Rather, it does not want them to get in the way of the things "on the left": individuals' interactions, working software, customer collaboration or responding to change. Tensions between Agile and UCD are due to the *extent* to which interpreters of the manifesto value items on the left more than the items on the right. The manifesto takes no position on what this extent should be, and therefore tensions between Agile and UCD are at best only partially due to the manifesto, but are mostly a question of interpretation and degree.

Agile's potential for sidelining UCD practices is thus not due to the manifesto as written, but instead to further turning the left hand side of the four clauses into crude rules and then banning the values on the right, e.g.:

1. Just have individuals and interactions: you do not need processes and tools.
2. Just make working software and no documentation.
3. Just collaborate with your customer: you do not need to negotiate a contract.
4. Just keep responding to change: you do not need to follow a plan

Some interpretations may thus devalue UCD to the point of its suppression, but this does not apply to all twelve agile principles that complement the manifesto [11]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Over half of the principles above should not block good support for UCD (i.e., Principles 2, 4, 5, 8, 9, 11 and 12). Conflicts however could arise from Principles 1, 3, 6, 7 and 10. Principles 1, 3 and 10 could all curtail UCD work, and perhaps eliminate it ('Early', 'shorter', 'work not done'). Principle 6 devalues documentation, which UCD uses to evidence its scientific values ("what do you mean by documentation?", Chap. 6). Principle 7 potentially conflicts with Principle 1: 'working' could just mean 'running' and not 'valuable' 'for the customer's competitive advantage'. Chapter 5 notes Agile's focus on functional requirements in relation to this.

A further problem arises if the only values considered in software development are only ones *interpreted* to come from the Agile Manifesto or its principles. Core UCD values and principles appear in neither. Agile values take precedence, whether these are orthodox or imagined, and thus Chap. 4 (proactive use of Scrum in a research and development setting) reports that the easiest and fastest to implement features would get implemented first and then demonstrated without prior user testing. Chapter 6 (imposition of Scrum on a university Participative Design project) reports a project team's dissatisfaction with the impact of Agile on its user community, who were effectively replaced as the customer by the project funder. Even so, integrating UCD with Agile is most challenging when UCD cannot adjust to Agile values. For example, Chap. 4 reports difficulties of estimating the time required for UCD work. Chapter 6 reports that developers could prototype faster than UX specialists, who were slowed down by the volume of community feedback. In fights between 'faster' and 'better', 'faster' won. The highest priority of Agile is "to satisfy the customer through early and continuous delivery of valuable software", and therefore 'better' should win. Unfortunately, Agile in practice can value speed more than value!

Comprehensive, effective and reliable integration will require relaxation of some of both UCD and Agile values some of the time. Agile holds up a valuable mirror to UCD, which must deal with valid issues that predate Agile (Chaps. 2, 4, 5, 6 and 7): UCD methods can lack transparency, be very time consuming, and result in inconsistent outcomes. For example, users can ask for different things (Chap. 8), which may be incompatible or irrelevant (Chap. 2), and UX specialists can be wrong (Chap. 7). One strength of this book's chapters is that authors recognize the need to rethink some key UCD practices and are willing to adapt them to make

their use possible in Agile settings. For example, in Chap. 3 (UX training courses and materials for developers) data analysis requirements for Contextual Inquiry are relaxed, and iterative development of training and developer support materials, resulted in time spent on A/B testing being halved.

Agile can make UCD better, and vice versa. Kuusinen's novel BoB framework for Agile UCD (Chap. 9) relaxes both UCD and Agile values, as is clear in the guidelines for using BoB (Table 9.3): designs need to be good and complete enough to be tested; early user feedback is better than late perfection; user interfaces will need to be refactored; and user expectations will need to be managed. Kuusinen's novel BoB framework also shows how UCD can make Agile better: 'working software' can be a clickable prototype, a working user interface with no functional back-end, a full working prototype, or a fielded version in use. UCD can thus hold up a valuable mirror to Agile, and a spirit of compromise can make both faster and better able to react effectively and wisely to change, build common understandings, and increase communication. Similarly, Chap. 2 balances positive attitudes towards UCD with the recognition that, even in companies with strong user-centred cultures, UCD needs to change. UCD is not seen as curretly mature, because optimal combination of methods is still not well understood. UCD is seen as addressing real users' needs and wants, but not necessarily with clarity on how users should actually be involved and how that fits into established development processes. UCD is positioned as an evolving normative ideological practice, with a primary focus on users, which is similar to Agile's focus on customers. Chapter 2's authors view process as a secondary concern in Agile and UCD, and Agile is seen to better promote a culture of self-evaluation. However, UCD is not the sole source of user-focused practices: good contributions can be made using management research on innovation, as well as established brand evaluation practices such as Net Promoter scores. There are thus important values and associated practices beyond UCD and Agile.

1.2.1.2 Values Other than Agile and UCD

There is more to software development than UCD and software implementation. Values from Agile and UCD are not the only ones encountered in this book. Chapter 6 (imposition of Scrum on Participative Design project) rightly argues that some tensions between Agile and UCD have to be resolved at organizational level, where decisions have to be made between conflicting values. There are also values of importance to specific application domains and markets that are supportive of Agile UCD integration. For example, organizations in Chap. 3 (UX training for agile developers) value:

- innovation from simple systematic consumer insights
- safety
- security

Chapters 2, 4, 5 and 7 contain further examples of companies' values including:

- experimentation (one SME had experience of both waterfall and PD practices)
- inclusive workplaces
- strongly phased hardware engineering practices, with early development of wireframes and systematic testing
- close collaboration with customers and suppliers, supported by transparent continuous communication
- high quality UX
- well-timed testing of feature groups (and not just what the last sprint delivered)
- democratic design driven by user-focus
- coherent synergetic product and service portfolios
- competitive advantage
- market intelligence, including price guidance

Interestingly, the Chap. 3 values of safety and security, as well as performance and accessibility, can get foregrounded to gain approval for what are really UX fixes (Chap. 10). Such 'passing off' is less common in organisations that develop safety critical and consumer products, with their associated 'testing' and 'insight' cultures. Every case study introduces domain values that favour some forms of integration over others for UCD and Agile. Mass market companies in Chaps. 2 (SME use of Agile) and 3 (UX training for developers) need high levels of confidence in UX before they release new products to market (one has a usability laboratory). Healthcare equipment in Chap. 3 is subject to usability and other requirements from the U.S. Food and Drug Administration (hence use of A/B testing). Projects in Chaps. 2 and 6 benefitted from Living Labs. Hardware produced in Chaps. 2, 3 and 7 organisations cannot be iterated or incremented as easily as pure software products or services. Research and Development (R&D) contexts in Chaps. 4 (proactive UX R&D use of Scrum) and 6 (imposed use of Scrum) allow more flexible self organisation than many other settings. In Chap. 4, only an R&D prototype was needed as a deliverable. Chapter 6's educational setting enabled open source practices, bonds of community, interns, student evaluations and other important participative practices. However, the funder's values introduced hierarchy and formality that limited the extent and form of participation. The project management team retained overall control and steered the direction of the project, appointing "champions" for each of the eight apps developed. In Chap. 7, agile teams had value expectations for enterprise software users that turned out to be incorrect: users' values were not predominantly instrumental. In short, each development context brought values that shaped both Agile and UCD practices.

The Smart Campus research project (Chap. 8) aimed to develop mobile apps to support a student community. A pervasive PD approach was expected, supported by multiple feedback channels. Before Scrum was introduced, activities completed had included community engagement and workshops, focus groups, diary studies, online ethnography, benchmarking university mobile apps, conceptual design, service infrastructure, research on online student communities, personas, scenarios, storyboards and sketches; in short, extensive upfront work preceded application

development. However, the unanticipated introduction of Scrum obstructed further PD practices. To understand the difficulties encountered, and how to overcome them, two interview studies were planned and carried out by a researcher external to the project. A literature review in conjunction with the interview studies revealed how Agile can obstruct UCD work practices; the authors benefitted from two 2014 literature surveys that covered 76 [12] and 71 [1] papers, which highlight the topicality of the chapters in this book.

These difficulties, which may obstruct integration of UCD and Agile when the 'user' is actually a community and is therefore not uniquely defined, contributed to the project not being fully appreciated by university management, even though the eight mobile apps were used and positively evaluated by the users: in fact, two of these apps had been developed by the student community, and there were 2000+ posts in the online project forum.

The case study chapters are particularly valuable in providing examples of organizational culture and values in context. In addition to the two sets of Agile and UCD values, a third set of organisational values must underpin Agile and UCD values to let them be acted on. Chapter 2's SMEs' practices do more than simply combine Agile and UCD: much broader ranges of values are involved, including those underlying business inputs to product and service development such as innovation and marketing [13, 14]. Chap. 2 also describes how user support and marketing contribute evaluation data and ideas for improvement. Similarly, success for Chap. 3's companies also involves more than Agile and UCD values, which may have to be relaxed to enable success. Chapter 2 thus suggests that UCD may have to relax on its scientific values in fast-paced market-driven environments.

Equally, organizational values may impede Agile. To adopt Agile, an organization has to value: trusting employees; self-organisation; continuous reflection and learning and being able to adapt when necessary. Many organisations have hierarchical cultures that are driven by top-down management. These cultures find it hard to adopt Agile until they embrace agile culture beyond their IT function.

Chapters 4 and 5 (first uses of Scrum in UCD contexts) demonstrate how well-motivated teams can resolve some tensions between Agile and UCD bottom up. As well as Agile, UCD and organizational values, there are values of importance to development team members, who are the source of a fourth relevant set of values. Chapters 5 and 6 recognise the contributions of established SE work and associated values [15–18]. Some of these values may be shared with employers, but others are professional values specific to roles, or reflect personal goals related to current achievements and aspirations. Examples include:

- meeting deadlines
- technical achievement
- internal software qualities such as maintainability, reliability, portability, efficiency and security
- professional distinctiveness
- creative exploration through trial and error: knowingly playing a 'game of chance'

- aesthetics, both visual and technical (e.g., beautiful code [6]).
- confident and personally secure integration of UX tasks within development
- highly directive prescriptive methods
- open flexible methods that maximize use of personal expertise

Chapter 7 correlated development teams' overall assessment of UX with their ratings on particular UX dimensions for software that they had developed in Agile UCD contexts. Significant correlations between their overall rating of UX and their specific ratings for 'good', 'desirable', 'innovative' and 'recommendable' indicated that development teams saw these attributes as major determinants of UX quality. These attributes correspond to market oriented values that align developers with their companies, valuing brand equity and competitive advantage. They are not primarily UCD values.

An important developer value is seen in personal responsibility for the consequences of design decisions. Chapters 3, 4, 5 and 6 all refer to Product Owners or project managers either generously adding to user sourced requirements (as advocated in Chap. 11), or more often overriding these. Not everything that customers or users want is desirable or possible. As feedback channels mushroom, it is simply not possible to take every customer or user suggestion on board, especially when many conflict.

Designers are a fifth source of values for development projects. Their designerly values are the sole focus of Chap. 11. Chapter 6 also advocates Design Thinking [19, 20] and thus designers' professional values (e.g. keeping up to date with Android look and feel guidance). Interestingly Chap. 6 is the only one to celebrate success, despite the funder having issues with the project: its Smart Campus apps were popular with users. In contrast, Chap. 7 displays UCD's ancestral distrust of designers [21], and disapproves of "designers' creativity and originality and … ability to break design conventions" [22]. Breaches of "design conventions" are seen as bad design decisions, when often conventions do need to be broken, as evidenced in recent debates on 'hamburger menus' for apps [23]. Similarly, Chap. 9 argues that "good UX should not be a game of chance, but it requires deliberate work," reflecting UCD's initial engineering values.

Agile comes close to creative practices, and thus has less overt distrust of creative designers than UCD. Chapter 11 presents a broad overview of creative design and notes how Agile has embraced some creative practices, but remains constrained by some engineering ideals. For example, although requirements can change, Scrum requires them to be clear at the start of a sprint [24]. Even so, Agile has softened up UCD to accept the creative design characteristic of problem-solution space co-evolution when integration with Agile is achieved. Chapter 9 thus accepts that we cannot know in advance what a system under development should be like and how it should be implemented [25], motivating a fundamental Agile principle of welcoming late change [11, 26].

Designers' creative inspiration is never completely down to Chap. 9's "game of chance", but creative designers expose themselves to influences in ways that cannot be completely deliberate. Often these influences, inspirations and opportunities arise

from their own work, which unexpectedly 'talks back' to them (Chap. 11). It is important here to distinguish designers' craft mastery from their creative inventiveness. The former is the material expression of ideas from the latter. Craft mastery can be exercised at will as required, but ideas that are repressed or suppressed may be lost forever. Historically, UCD has posed more of a threat to creative designers' values than Agile has, although both embrace idealized engineering design values that can repress or suppress creative practices. However, Kuusinen's 'design debt' in Chap. 9 reasonably expects designers to relax the production values associated with their craft mastery to allow rapid deployment and testing.

Chapters 4, 6 and 8 accept the uncertainties and generous opportunities of creative design. Chapter 6 refers to [27], which describes a 1-day design studio that was used to bridge between UCD and Agile. This was a sensible compromise a decade ago, but has now been superseded by Google's week long *Design Sprints* [28], which can provide a firm foundation for Agile UCD at the point of project inception or within upfront activities. As with all similar innovation approaches (including Design Thinking [19, 20]), Design Sprints combine design, UCD, technology and business perspectives in a single process where, with skill and will, these can be balanced against each other. A similar mix of technical, art and design perspectives was advocated in a position paper [6].

Customers are a sixth source of project values. Customers' values in Chaps. 2, 3 and 5 drive the integration of UCD and Agile. In Chap. 5, an Agile UCD methodology was co-designed between a software developer and a university. Otherwise, customers are given limited detailed attention in the case studies, as are users' values. Chapter 8 reports concerns that were raised at the workshop about UCD values being overlooked in the commissioning process.

Users are a seventh source of project values. If UCD goes to plan, users' needs and wants should always be understood and responded to. However, Agile contexts are forcing UCD to reconsider the ideal that if users have a need, then software must meet it. A focus on the user at all times must relax to consider other equally important considerations from business and engineering. Also, the ability of UX specialists to know users' priorities, even when they have worked with them on a project, is called into question in Chap. 7, where users were asked to rate overall and specific UX dimensions for products developed by Agile teams. Associated project teams were also asked to rate their product for overall and specific UX, but from both their own and from their users' perspective. Overall ranking of UX dimensions was not possible (i.e., how users and teams valued some dimensions more than others), since the products differed. However, comparisons between each specific UX dimension and each product's overall UX rating did reveal significant correlations, but these differed for each comparison (users, teams' own, teams' predicted). These differences indicate that that users' value systems are not what Agile teams think they are. For enterprise software, instrumental dimensions should logically better predict overall UX, but this was not so. Analysis by role indicated that no team roles were particularly accurate, although POs and UX Specialists (UXS) could better predict user ratings than developers. Commitment to UCD values does thus not translate automatically into accurate user empathy.

At the workshop, a danger was noted that HCI knowledge can be seen as 'common sense' that everyone knows. HCI expertise needs to be appropriately valued, otherwise customer or user preferences could take precedence over specialist HCI knowledge that indicates better design options. Users and customers should not always be given what they want. Thus in Chap. 3, requests for fully completed example templates were not acted on. It is important to bridge across (and adjudicate between) the cultural values that rendezvous in digital product and service development.

Agile and UCD values are thus not the only ones in play when integrating their practices and associated cultures. Instead, integration always occurs in contexts where five further sources of values are in place, i.e., organizational values, which mediate across four specific role values for developers, creative designers, customers and users. Cultures manifest lived values through material and social practices (e.g., documentation and meetings respectively). However, in Chap. 2, the individual organizational cultures are not homogeneous. Different roles within organisations bring different values with them, and in the largest SME "heated conversations can happen". If these are well managed, design quality will benefit.

While values may appear to be logically incompatible, cultures only actually clash through incompatible material and social practices. Thus claims made for irreconcilable differences between Agile and UCD cultures have to be evidenced by practices and not values. Such practices are shaped by methodologies and their associated team and task structures.

1.2.1.3 Cultures and Methodologies

Workplaces and projects are focal points for the convergence of scores of values from at least seven distinct sources. Only two are methodologies, which Chap. 2 states are normative and ideological, i.e., they are primarily prescriptions supported by descriptions of required practices. Methodologies signal what matters to practitioners, either directly through manifestos or indirectly via their practices. Ideologies thus become inscribed in people, places and things, e.g., respectively through Scrum's roles, stand up meetings or Scrum Boards. Work practices, environments and artefacts need to be compatible with a methodology's values, but methodologies in turn must be compatible with organizational and project roles' values.

Cultures are complex. Simplistic shoot outs between Agile and UCD ignore other sources of values, to which methodological ideologies are subordinate. Important enablers and obstacles lie outside of Agile and UCD, within the organisations that work on integrating Agile and UCD. Many current organizational enablers increasingly favour Agile, hence its fashionability (Chap. 10). The maturity of much IT has changed its role from a 'service' function to being the 'driving' function of business. New business ideas often only come into being when they are realized through IT. Business and IT are thus merging, giving rise to challenges for organisations that vary in magnitude. For some organisations, the challenges

are substantial, and major difficulties with managing these challenges impede successful adoption and integration of Agile and UCD. At the same time Agile has been widely embraced because it addresses these challenge for business. In many ways Agile isn't just a methodology. Instead, many of its concepts are enablers for organisational transformation through new ways of configuring 'work' in modern businesses. Agile can be applied in many different settings. This book's case studies span organisations at different stages of maturity in terms of their readiness to exploit Agile UCD.

Agile methodologies have undoubtedly become very dominant recently. As IT has transformed from a 'service' function to the 'driving' function of business, IT development has grown from 'one off projects' to continuously developed 'online products'. Agile approaches and concepts have helped IT departments to change their approach in order to deliver what businesses need. However, as Agile is adopted, it is not just specific practices that need to change, but the wider cultures within which practices occur. Such changes can be difficult, especially in large organisations.

The case study chapters provide a broad understanding of culture, with balanced accounts of how organizational and professional cultures combine to support integration of Agile and UCD. While not perfect, and thus potentially disappointing to UCD and PD (Participative Design) ambitions, the quality of integration is at least reasonable, and organisations are finding ways to make further progress, in contrast to the workshop's "challenges and obstacles" focus on culture.

Cultures are not deterministic, nor do value preferences bestow competences. For example, a Chap. 2 SME valued home-based testing, but had no role in place with responsibilities for engagement and liaison with test households. Nor had this SME worked out how much contact developers should have with users. Values can only be lived through material and social practices. Until these are in place, values will remain frustrating aspirations.

The case studies are particularly valuable in demonstrating that a methodology's values cannot completely constrain software development. While values may be clearly advocated in manifestos and seminal publications, their impact plays out in complex dynamic contexts that can balance one set of values against another, and Agile values do not always win (either orthodox or imagined). Scrum is by far the most popular agile methodology currently [29]. However, all of the 'Scrumish' practices in this book's case studies breach key Scrum rules. There is extensive up front activity, breaches of the closed window rule [4], tasks running across sprints, involvement in sprint planning beyond the core development team (PO, Scrum Master, developers), many roles that do not exist in Scrum, stand up meetings involving roles outside of the core development team, and missing sprint reviews and retrospectives. For example, despite Scrum's preference for face to face meetings, extensive electronic documentation and project records are common (Chaps. 2, 4, 5 and 6). Electronic documentation in a software tool can be rapidly accessed and updated, reducing some drawbacks of documentation assumed by the Agile manifesto.

Much of the dogma associated with Scrum has been neutralized in this book's case studies. This corroborates Meyer's view that development teams have to interpret and adapt Agile, and that they sensibly do so in ways that avoids dogmatic damage [4]. For example, until fairly recently, Scrum distinguished the 'committed pigs' in the core development team from 'involved chickens' of customers, users and other outsiders, but this has now been relaxed [30]. In Chaps. 2, 4, 5, 6 and 9, decision making is shared by the core team (PO, Scrum Master, team members) of developers with other roles, avoiding the restrictions of Scrum dogma.

The ability to adapt agile methodology can be attributed to pervasive values of agile culture such as self-organisation, continuous reflection and learning, and methodology adaptation. All the case studies contain examples of how agile teams have adapted their approach to achieve their aims, exemplifying the strengths of agile culture that may indicate why Agile has become so popular and fashionable.

The following positions on culture and Agile UCD thus emerge from the workshop and the chapters:

- UCD and Agile's values are largely disjoint but not inherently incompatible. Dogmatic rules from either side are more likely to be a source of tensions. Truly flexible and collaborative responses however can minimize tensions.
- Systems development takes place in a large soup of values. Agile and UCD only contribute some of the values in play. Most come from other organizational and professional sources (e.g., marketing, creative designers, customer support, security and safety experts). These may be disjoint sources of either conflict or synergies. In principle, UCD should consider user values, and Agile should consider customer values. Even so, specific values from each cannot guide UCD or Agile work until they are understood and acted on (as above, value preferences do not bestow competences). User and customer values are best understood by giving them their own place in the 'soup of values', not by being represented indirectly by UCD or Agile.
- Conflicts and synergies are manifested in teams and tasks in work contexts, where the ideologies of methodologies meet the realities of software development work.

Methods require knowledge, orientations and expertise for effective application. Development teams must collectively have appropriate values, knowledge and expertise. In practice, methodologies shape development work indirectly via:

- Divisions of labour as required roles and their associated responsibilities within development *teams*
- Development *tasks*, via prescribed and proscribed practices.

These two further themes, teams and tasks, are next discussed. What makes teams and tasks 'good' relates to values associated with Agile UCD in specific contexts. Teams on tasks deliver on values, hence the workshop focus on fitting together the big picture of agile theory and methods with lower levels detail of day-to-day practices.

1.2.2 Software Development Teams

Team work requires a division of labour and some form of leadership. Both are usually supported by named roles with specified responsibilities. Role names may not fully reflect or even match responsibilities. Responsibilities may arise that existing team members are poorly equipped to deal with. This may be due to disciplinary or professional background, or to relative inexperience. To thrive, all organisations must learn, and thus teams have dynamic aspects that must be considered alongside static organistional structures. Key themes in the workshop and chapters related to teams that will now be discussed are:

- Roles and responsibilities
- Team boundaries and communication practices
- Cross functional capabilities of teams and individuals

1.2.2.1 Roles and Responsibilities

The third largest group of post-its at the workshop was "Roles and people" (Chap. 8). A people focus was seen as key to addressing integration challenges for Agile UCD. The most popular agile method [29], Scrum, allows only three roles: Product Owner (PO), Scrum Master and multiple team members [4]. There must be no differentiation of rank or expertise within team members. They should all be capable of carrying out all development tasks. Such simple teams never appear in any chapter. For example, Chap. 2 has multiple roles across its three studies:

> Sole/multiple differentiated POs, social media manager, support team, developer, customer lab, independent tester, functional tester, design and verification and testing, quality management, user contact, feedback categorizer, feedback manager, user, market analyst, coordinator & project manager, creative director, senior art director, customer.

Chapter 2 thus reveals the diversity of successful Agile UCD practices, with each SME exploiting a range of roles and practices, especially in relation to customer and user feedback. One SME had a dedicated social media management group, evidencing their strong commitment to user and customer feedback. Although all used 'Scrum like' methodologies, these were significantly extended and complemented with additional roles and externally facing activities. One organization collaborated with third party suppliers, extending product development teams beyond, for example, Scrum's preferred roles of developers, Scrum Master and Product Owner (PO). All three SMEs had mass market orientations that create strong user-focused cultures.

The 'Scrumish' nature of most agile development (Chap. 10, i.e., not fully compliant orthodox Scrum) is clearly indicated by an order of magnitude difference in role count across case studies. Chaps. 3, 4, 5, 6, 7, 8, 9 and 10 add to Chap. 2 roles, e.g.:

> mechanical engineer, proxy product owner, senior researcher, graphic designer, customer committee member, student, intern, architect, user experience specialist (UXS).

The existence of further roles such as junior researcher and hardware engineer can be inferred. Interestingly, while Scrum has only three roles, the DSDM agile methodology, has a dozen [31]. However, only four appear above (some in specialized forms).

In several case studies, the PO role was filled by either an existing role (e.g., UXS, project manager), a pair of roles, or a group (Chap. 5's customer committee comprises a PO, a development team member and two customer representatives, ideally one user and one business representative). Role complexity partially reflects the size and maturity of the organization, but as roles increase, teams become more clearly cross-functional. As team size increases, the need to split into subteams with leaders increases. Role complexity also reflects skill specialization.

Agile methodologies rarely distinguish users from customers, but every case study does. Chapter 6's Participative Design (PD) context required the student community to be the customer, but the university as project funder effectively took this role. Customer-user distinctions vary between software vendors (who develop then sell) and software development services (who are paid to develop). For the latter, customers may limit or prevent access to *their* users. Also, stakeholder roles beyond users and customers come into consideration for specific projects (Chap. 4 thus writes not of UCD but HCD – Human-Centred Design).

Clear effective positions, policies and practices are needed with regard to roles, responsibilities and relationships of customers and users in Agile UCD. The issues associated with roles are largely to do with their adequacy, which in turn depends on the responsibilities for each role, and how they can communicate with each other. In the case studies, if a role was needed, it had usually been created.

UCD values are relevant to UXS, PO, Scrum Master, developer, customer and user roles, but other sets of values take precedence for other roles. The range of roles associated with software development reflects the spread of values within the organization. The appropriate division of labour across roles is contextual. There can be no universal optimum, nor will Agile or UCD values always dominate. Chapter 2 thus argues against fixed roles. Instead, a full set of roles needs to be considered holistically on case by case basis for each organization and/or project. Holistic considerations of the division of labour are also important to ensure that someone is responsible for UX overall (Chap. 10). This is clearly important in relation to UCD values.

The existence and names of roles impacts less on development work than do actual responsibilities and the ability to discharge them, and power and a lack of responsibility. Responsibilities can be sources of tensions and controversy. Professionals may not have only one role on only one project, but several across several projects. This is especially so for UXSs in a central team, as opposed to a dedicated UXS per project. Similarly, Chap. 4 argues that Scrum cannot work well with part-time staff who are not fully allocated to a single project. Scrum Masters found it difficult to ensure that projects were appropriately resourced as staff were (partially) reallocated to other projects. Part-time team members are not always on site and available for stand up and other meetings. Chapter 6 also noted a similar problem: differences in shift work meant that some interns and students could not attend regular face to face meetings.

In Chaps. 5 and 9, people are allocated to both tasks and roles. Tasks may be shared, especially by a pair of a developer and a UXS. Figure 9.4 in Chap. 9 proposes sharing of responsibilities across four roles, with all roles being responsible for product vision and understanding the user. This requires developers to be involved in UCD/UX tasks, and preferably to have some contact and interaction with users. Chapter 9's BoB Framework supports its flexible division of labour with guidelines for people (e.g., co-operate, respect others) and tasks (e.g., task allocation should respect professional expertise). There is clear recognition here of the diversity of values in play for software development, and the need to support the right values at the right times.

Some degree of collective responsibility is preferable. The resulting management challenges should be worth it. In Chaps. 4 and 5, teams decided to proactively apply Agile and took a bottom-up collaborative approach to Agile UCD. Within their emergent processes, role specific responsibilities were added to those for pairs and teams. For example, POs added to requirements when they saw gaps in elicited user and customer needs and preferences. In Chaps. 2 and 6 (as part of the customer committee), POs had to exercise judgement, and took responsibility for avoiding neglect and resolving conflict in the face of incomplete and conflicting user requirements. Avoiding neglect is an example of the creative design principle of *generosity* (Chap. 11), with customers and users receiving more than they envisage. Rigid UCD that restricts software capabilities and qualities to what customers and users ask for can obstruct high quality design work.

POs thus shared responsibilities with customers (as Agile requires) and users. This is an example of a more common phenomenon, where the scope of the PO's role expands and contracts on a case by case basis as a result of internal and external consultation. Cumulatively, this can make it almost impossible for POs to live up to their responsibilities, especially within the constraints of Scrum. Chapter 2 presents examples of breaking these constraints, with positive outcomes, through multiple POs and through customer involvements in thematic focus groups, where they fed back on quality in use and internal software quality. Similarly, Chap. 5 describes the responsibilities of a customer committee that includes the PO.

In Scrum, the PO should represent the customer. However, customers work directly as project team members in some case studies: customers as funders were directly involved in top down decision making in Chaps. 2 and 6, which constrained the autonomy of the PO and the development team. In Chap. 2, customers organized beta testing and also carried out user testing independently of the development team (Chap. 10 also refers to user testing by customers). The design, verification and testing role for a SME in Chap. 2 organized the user panel and user testing, and also filtered problem reports in the Bugzilla tool.

Developers' responsibilities were in focus at the workshop. Agile empowers developers. UCD's longstanding distrust of designers and developers [21, 32] was sometimes in evidence at the workshop. For example, bias was a concern when developers are responsible for evaluating the quality of their UX work. At the same time, measures to improve developers' UX capabilities attracted much interest, with 20 post-its for the 'developers doing UCD' affinity group being the largest group

within ‘interesting points’. In Chap. 3, ‘developers doing UCD’ is supported by a training role, with responsibility for deep learning, maintaining the requirement for learning by doing, and rejecting requests for detailed examples that could undermine this.

Developers need training and support to take on UCD work. Chapter 6’s PD context created a high volume of user feedback. Project management initially reviewed and prioritized user feedback, but left it to the development team after the move to Agile. Developers would proactively review postings on the project forum, and later the Github tool, when they needed to, but did not feel responsible for letting users know whether and how their feedback was being acted on. Developers did not have time for this. However, users wanted more transparency through feedback about their suggestions. PD advocates felt unease about not taking needs from the student community fully into account.

Training can also determine how roles are allocated. The Scrum Master role in Chap. 6 was filled by the project’s Chief Technology Officer (CTO), who read up on Scrum and introduced it to the development team through a sample Scrum planning meeting.

Managing user feedback is a common issue across this book’s chapters. Chapter 2 presented the very wide range of channels that are now possible for user feedback, and the challenges in creating and allocating associated roles. For example, there were uncertainties over the need for, and nature of, a new household testing role in Chap. 2. Chapter 6 draws on the PD literature to contrast informative, consultative and participative forms of user involvement. The PD focus in Chap. 6 led to conflict here, with researchers and educators wanting their technically skilled computing students to be participants, but developers and management wanting them to be mostly informative. Student users themselves were mostly informative, with only 27 % of forum comments suggesting improvements rather than simply reporting problems (67 %). Students thus mostly became app testers, rather than sources of requirements for apps.

A lack of responsibility or power is the main challenge for UCD roles, especially when other roles cannot or do not pick up uncovered UX responsibilities. Gaps here are often reflected in team boundaries and communication practices.

1.2.2.2 Team Boundaries and Communication Practices

Team work requires co-ordination and collaboration, and both require communication. The second largest post-it group at the workshop was “Teams and Co-ordination”, indicating its importance. Communication is an important focus in Chaps. 2 and 6.

Roles who can fully meet their responsibilities on their own are not part of a team. Teams who can fully meet their responsibilities on their own are working for themselves. Communication, collaboration and co-ordination are thus required for effective software work, since team work requires collaboration and co-ordination and customers require communication. Developing software for customers requires

interactions with stakeholders who may not be part of the project team, which may correspond to the development team, or extend beyond them. Some roles can feel excluded, especially UXS roles who cannot participate in routine development team meetings such as daily standups (Chap. 10). At the workshop, it was remarked that UXS roles can be seen solely as a service team for research (before development starts) and analysis or evaluation (after an iteration). This may not involve UXSs enough, as it excludes them from interaction design and may also limit their access to the PO (as noted at the workshop, Chap. 8). Furthermore, it restricts UXS support for problem-solution co-evolution, a key characteristic of creative design (Chap. 11), since UX expertise is excluded from framing problems and solutions within sprint planning, progression and retrospectives. Pair designing can avoid this, with a UXS working closely with a developer during a sprint (recommended in Chap. 9, and practiced effectively in Chaps. 6 and 10 projects).

Teams may nest like Russian dolls, i.e., an agile development team within a project team within a virtual organization of stakeholders. Team membership can vary. For example, in Chap. 2, a social media manager and UXSs from a 'customer lab.' would be invited as needed to sprint meetings (adding to the responsibilities of the PO or Scrum Master). Also in Chap. 2, one PO decides on a case by case basis how to involve external partners who are part of an extended development team. The resulting virtual organization requires upfront design and subsequent co-ordination.

Chapter 9 argues that the whole team should be involved in user communications, which requires feedback channels to be in place. While this could mean a broad project team or a development team within it, the intention is that developers are involved. Chapter 10 observes that developers can be 'protected' from customers, which becomes difficult when developers access communication tools that let customers and users contribute comments. In Chap. 6, some developers chose to ignore some "kinds of users" based on their online behaviors. Even so, the online forum for Smart Campus did host some good discussions and created a mature sense of ownership. Some workshop participants felt that disagreements that are managed openly and inclusively can result in better design decisions and more effective team working (Chaps. 2 and 8). An alternative approach, also in Chap. 2, frames open fora as 'users helping users', lowering expectations for the SME to make active contributions (while still being able to monitor comments). However, this is specific to a single user feedback channel, and would not generalize to all project communication.

The extent and nature of developer contact with customers remained unresolved in one of the Chap. 2 case studies, where hearing opinions and problems was thought to be useful for forming understandings of users, but has "to be channelled in some way", which is likely to require some form of tool.

Agile's preference for face to face communication is not met in most of the case studies. While the Scrum Islands in Chap. 5 are so continuously face to face that they remove the need for daily stand ups and sprint reviews, multiple asynchronous online communication channels were nevertheless in use for this case study. For example, the PO used the Redmine tool to formalise backlogs, classify issues and assign them to development team members. In Chap. 6, remote workers made use of

dropbox for sharing documents, and Google Hangouts for design discussions. Chap. 2 (Table 2.3) lists dozens of communication channels and associated media (e.g., face to face, documents, online). The quality of communication channels, and of access to them, can either enable or obstruct roles in meeting their responsibilities. In Chap. 6, only the Scrum Master, who was the Chief Technology Officer (CTO), could write into the developer wiki, thereby creating a hierarchy that goes against Scrum principles. Also, it was noted in the workshop (Chap. 8) that customers may not be used to online communication but send feedback through 'snail mail'.

Although Chap. 4 concludes that Agile requires full time team members, they developed an innovative two sided Scrum Board, with inventive use of coloured post-its of different sizes, to involve and acknowledge part-time team members. A planning whiteboard and display space on two walls further communicated the big picture, supporting synchronisation and making visible the work of non-Scrum roles such as UXS and Architect. However, such uses of physical space require routine co-location (as do Chap. 5's Scrum Islands and Chap. 9's BoB framework).

Chapter 3 advocates training developers to give them a common language as a foundation for aligning design and development within sprints, but adequate tool support is needed to support communication across all stakeholders. Chapter 6 reports problems with communication within a PD project, where the lack of notifications in its online forum reduced developer feedback to users. Along with users' initial inability to track problem tickets, this lowered expectations that did not improve once the use of GitHub did make problem tickets visible. Also, developers could not easily search GitHub.

Online tools for communication with users and customers are becoming vital to effective development of digital products and services. A quality infrastructure needs to be in place before any development sprint releases a version to customers. This is not an area where difficulties can simply be refactored out in a later sprint. Once created, low expectations are hard to undo. Poor PD infrastructure in Chap. 6 also made the volume of content from the multiple feedback channels difficult to manage, slowing down design work and making it very difficult for designers to work one sprint ahead of developers. Formal, semi-formal and informal communication approaches were experimented with, but none proved to be fully satisfactory for integrating design and development.

The range of communication tools and media across the case studies indicates the impact of Web 2.0 and its social media on Agile UCD. Workshop attendees were struck by the range of media and the purposes that tools were used for. Such tools and media can bring companies much closer to their customers and users, but exploiting opportunities here is not straightforward. At the workshop, integration and consolidation of different needs and desires was identified as a challenge. The volume of content can become a torrent. Current tools do not always provide good support for searching, filtering or responding. A SME in Chap. 2 has developed its own review aggregation tool. It is important to frame expectations for users and customers to avoid disappointment or frustration. As tools become more capable, users can expect more from them.

Online tools are also used across the case studies within development teams, despite the Agile preference for face to face meetings. One can wonder what a new Agile Manifesto for 2016 would look like, 15 years after the original [11]. Many teams are not co-located and cannot be, due to part-time or shift working on projects, or essential cross-site development, sometimes global and 24/7.

Agile UCD thus depends on adequate access to a broad range of well managed inclusive communication channels across different media. Agile's preferred face to face meetings are thus extensively complemented by persistent physical and asynchronous online media. However, a team needs a shared language to use these effectively, especially as teams become more cross functional. The training in Chap. 3 aims to develop this, for teams with 4–5 years' experience of Scrum. Similarly, a SME in Chap. 2 "encourages everybody working on a project to constantly take a step back and actively try to view the product through the eyes of a customer as well as a user". The Art Director for this SME had a strong UCD orientation. This leads into the observation that the social practices associated with roles and team boundaries, and the material and social practices associated with communication, depend on specific individual capabilities, which are next briefly considered.

1.2.2.3 Cross Functional Capabilities of Teams and Individuals

One of Agile's strengths is its focus on learning, e.g., through Scrum's Sprint Retrospectives. Chaps. 4, 5 and 6 present clear evidence of experiment and invention (including independent adoption of Scrum by UXSs), although the unplanned imposition of Scrum in Chap. 6 (apart from the CTO attending a Scrum Master course) made it hard to fully resolve issues of user involvement, documentation and communication. The case study in Chap. 5 benefitted from the company's existing initiatives on Learning Organisation and Experience Factory Models, which had already established favourable support for learning before adopting Agile.

The workshop (Chap. 8) noted the importance of team building and support and facilitation for key design practices. This cannot be achieved by specialised roles alone. Instead T-Shaped People [19] are required, for example developers with design literacy or designers with coding literacy (workshop discussion of position papers for Chaps. 6 and 9). Chapter 9 notes a need for UXSs to understand large scale software development. POs must be as T-Shaped as possible (Chap. 2), but not impossibly so!

Team capabilities for Agile UCD need to be sustainable and scalable (Chap. 8). Continuous education and development, where people learn from each other, needs to be part of Agile UCD practices, resulting in a virtuous circle of learning (Chap. 9). The Scrum backlog concept could be extended to training here, with a training backlog running across projects.

The position papers for Chaps. 3 and 7 drew considerable comment at the workshop, with 16 % of "interesting issues" post-its focused on "Developers doing UCD", which was generally seen as "a very good idea" that attracted "great

interest", although one attendee wondered if there is "enough time for developers to also do UCD duty" and another felt that this form of Agile UCD integration would depend on company type. Although Agile has no overt focus on users, the case studies reported that developers were, or became, willing to carry out UCD work, but are initially held back by a lack of competence and thus confidence. Chapter 3 thus reports on the development of formal UCD training for developers that avoided 'observe and learn' education in favour of independent hands-on project specific cases within 3 months of the training. The pair development advocated and reported in Chaps. 6, 9 and 10 is an informal approach to building UX skills in developers. Both formal and informal approaches address the UX as bottleneck problem by providing additional UCD resource through developers.

In summary, Agile UCD needs adequate teams. Roles and communication practices matter, but the foundations for success lie in individuals' attitudes and capabilities.

1.2.3 Software Development Tasks

Much of the focus during the workshop was on fitting the big picture of agile theory and methods together with the lower level detail of day-to-day practices. This book's chapters present innovative practices that can make progress on remaining challenges for Agile UCD. While there is a fine line between how a development team is structured and how work is assigned and completed within the team, the lower level detail of day-to-day practices is primarily focused on development *tasks*, albeit within the context of organizational structures. Chapter 2 argues that issues concerning Agile UCD are more practical than conceptual and celebrates Agile's focus on learning. Sect. 1.2.1 above supports Chap. 2's position: there are few value conflicts and conceptual mismatches between Agile and UCD. There are gaps in each methodology, but as Chap. 5 argues, this lets Agile and UCD complement each other. The challenges lie in creating practices that integrate both methodologies, for example through the training regime in Chap. 3 or the BoB framework in Chap. 9.

Issues and ideas in chapters in this book related to tasks are discussed below at the levels of processes, iterations (e.g., sprints) and low level activities.

1.2.3.1 Process Level Issues and Ideas

In the most minimally extreme form of Scrum, there would only be sprints that implement user stories. While Scrum appears to be such a form, there are always activities prior to the first development sprint. The key questions for Agile UCD are what these upfront activities should be and how much time and resources should be allocated to them. Agile values favour minimal upfront activities [11], whereas UCD ones require comprehensive planning. Chapter 10 argues for a big picture being established from the outset through a thorough pre-study. However, UCD is not an

obvious part of Agile and can be deprioritized and passed over. At the same time, agile practices are flexible, with the case studies in this book providing extensive evidence that UCD can be integrated into Agile. Where there are difficulties, this can be due to UCD. For example, UCD costs and/or timescales can be difficult to estimate, which can make it difficult to systematically integrate UCD into Agile (Chap. 4). However, a lack of supporting documentation from initial contextual studies will compound this problem (Chap. 8).

The more specific process challenges for Agile UCD thus concern: upfront activities; transitions between sprints; types of sprints and their synchronization; and constraints on the overall process.

Upfront activities prior to the first development sprint are sometimes referred to as Sprint 0 [33, 34], but Chap. 5 also has an *Inception* stage prior to this. UCD tends to overlook Inception, where project sponsors develop an initial brief and vision. Much can already be in place as regards software and hardware platforms, key features, design purpose and target market at this point, but UCD is typically envisaged as starting with a clean sheet. UCD's 'no designing before users and tasks are thoroughly understood' rule [21] displays a remarkable ignorance of how business decisions on new products and services are made [35]. Both Agile and UCD have to situate themselves in the wider contexts of customers, designers, developers and organizational sponsors. Chapter 5 is unusual in acknowledging the Inception phase. Agile and UCD both need a good awareness of how and where projects really start, which is usually before any project team meets for the first time. Some agile approaches however do take these factors into consideration. DSDM [36] has a Pre-project phase (for the development of a proposal in line with strategic goals), a Feasibility phase (which looks at project viability and a high-level investigation of potential solutions, costs and timeframes) and a Foundations phase (to develop a high level view of how the project will meet business needs and who will be involved). RUP [37] has an Inception phase similar to DSDM's Foundations, during which the business case and high-level requirements are developed as well identifying people who will be involved.

Good initial plans for product development are important when responding to user feedback. In Chap. 2, an extra feature was regularly requested by users, but was not implemented because it "would make other, quite specific, long-term plans for the software impossible on a technical level."

Sprint 0 can take many forms, but it is seen as vital for establishing a baseline understanding of users and their needs. Chapter 9 also advocates design activities, with a clickable prototype as a final outcome of an early product definition workshop, with the product/service vision and the most critical user stories also established as an initial backlog. Agile UCD can build on DSDM's Workshop Facilitator role here [31, 36] to combine business, design and UCD perspectives. In Chap. 5, Sprint n.0 lasted 40 days (20 % of the elapsed budget at the time of writing). Chapter 9 advocates minimizing the time spent on upfront activities. However, what is minimal will depend on what is needed.

Insights developed during Sprint 0, and decisions made there and during inception, need to be preserved during the transition to the first development sprint,

and in all subsequent transitions between sprints. Chapter 9 notes that sprints can stop developers from thinking ahead, which carrying a common vision forward can support. The big picture from a common vision can support other activities such as chunking features within sprints to support UCD activities, or planning two or more sprints ahead (Chaps. 4 and 5). Both example activities here rely on being able to decompose the big picture into manageable chunks of work, an issue flagged at the workshop [6].

Transitions between iterations such as Scrum sprints also depend on the types of sprint in a development process. When completely focused on development, Agile typically has regular feature development and refactoring iterations, which address internal software quality [2] and add no new features. However, 'design refactoring' can address external software quality (Chap. 9). Also, the most common approach for Agile UCD is one sprint ahead [38, 39], which can add design and testing sprints alongside development ones (Chap. 9). This can result in synchronisation challenges, but there were only 5 (out of 145) post-its related to synchronization at the workshop (and a further 13 related to more general time issues). Also, the Chaps. 4 and 5 case studies worked (at least) one sprint ahead and reported no issues, but both were using Scrum for the first time in a mature UCD setting (as at Autodesk, where one sprint ahead originated [20, 21]).

Synchronisation challenges can be eased in two ways. Chapter 4 allowed bottomless sprints, which need the capability to look a few sprints ahead. Case studies in Chaps. 2 and 4 also let some UCD activities cut across sprints. User testing can be a source of difficulties here, as user tests may not fit easily within a sprint, and when they do, it can be hard to respond to test feedback before the end of the sprint. Even so, Chap. 10 notes that usability techniques can be fitted into Scrum [40]. In Chap. 3, Contextual Inquiry interviews were carried out one at a time, and thus there was no need for time consuming data consolidation. However, some challenges are not so easily overcome, for example, working across time zones (Chap. 4) or torrents of user feedback (Chap. 6): "one of the most disruptive elements in effective synchronization."

Process level issues are above the iteration level. Managing them well makes it possible to get the best out of each iteration. Exact planning is not possible in creative work (Chap. 11), so Agile must be flexible enough to accommodate this. Ad hoc interventions in Chap. 2 were noted at the workshop. Chapters 4 and 9 advocate chunking groups of tasks. Chapter 4 reports difficulties in accessing users for UCD work within the constraints of sprints. In response, a longer time frame was adopted, with a calendar showing availability of team members over the next 3 months. This made it possible to look ahead when planning future sprints.

1.2.3.2 Iteration Structure Issues and Ideas

Agile iterations such as Scrum's sprints are timeboxed development tasks for specific goals. The most straightforward way to integrate UCD and Agile is to incorporate the former into the day to day work of the latter, i.e., into the iteration

structure of an agile methodology. UCD difficulties with Scrum are often attributed to the length of sprints, but this book's Scrum case studies used a range of flexible sprint lengths, and ran some activities in parallel for more than one sprint ahead. It makes sense to fit as much as possible within the main sprint structure and backlog. In principle, all forms of development activity can have their own backlog (Chap. 11), or be combined into a single one (Chap. 9). The sketch wall in Chap. 4 could be regarded as an extension of the Scrum Board.

Chapter 8 refers to Agile's design/develop conundrum, i.e., an intricate difficult problem that has only a conjectural answer, in this case the question as to when to design and when to develop. Pair working (Chaps. 5, 6 and 9) can achieve day to day integration of both design and development. As well as reducing synchronization problems, it lets sprints vary their balance between designing and developing. For example, UCD activities can run one or more sprints ahead alongside code refactoring, or new feature implementation could pause to allow design refactoring in response to user or customer feedback. Both 'technical debt' and 'design debt' must be accepted (Chap. 9), i.e., leaving a feature or its UX at a (very) basic initial standard to allow more rapid deployment and customer/user feedback.

Chapter 5 reports a simplified sprint structure that required no end of sprint review or retrospective. This was made possible by a customer committee and Scrum Islands, with the latter providing continuous feedback and the former being able to continuously plan ahead. This should reduce concerns about rigid sprint structures as obstacles to UCD. Chapter 4 reports interesting restrictions on actions arising from sprint retrospectives that allowed manageable changes to be made to sprint practices within the overall process. The case study here was particularly innovative. Methods that did not work, such as Agile's Planning Poker were replaced with a simpler resource: different sized post-its on the scrum board.

Some work will not fit into a standard iteration, although every effort should be made to make this possible when needed, and to deliver working designs as part of this (Chap. 9). Where this is not possible, then developers may be forced to work ahead of supporting UCD work (Chap. 6). However, the lengths of iterations can vary, which can be to the benefit of UCD work. Case studies report ad hoc interventions that breach required Agile practice, e.g., relaxing the closed window rule for essential iterative development (e.g., Chap. 4, which also let developers work ahead of the current sprint), although the closed window rule is reported as a problem in Chap. 10.

Iterations can be planned more than one ahead. Chapter 4 planned 5 sprint milestones ahead (initially for 1 week sprints), which allowed UX work in advance, bringing UX and feature implementation as close as possible, but not absolutely in parallel. Availability of part-time team members had to be factored into planning.

1.2.3.3 Low Level Activity Level Issues and Ideas

While processes can span months and even years, and iterations weeks and sometimes months, activities within sprints may last only days or hours. It is at this level

of design, implementation and evaluation tasks that Agile UCD is actually delivered. Process and iteration structure may potentially enable or obstruct, but the enablers and blockers here only become real in the context of actual development work.

Tasks or activities are supported by *resources*, which may be grouped into *approaches* and named as re-usable methods [41]. Approaches are incomplete, and resources can be too. Approaches become methods through development work, which adds, completes and adapts resources. Both UCD and Agile methods and practices may have to be adapted for Agile UCD. Chapter 3 thus adapted three UCD methods through two to six iterations of training materials to fit its industrial development contexts. For example, contextual interview notes can be indexed using sequence models (contextual interviews and sequence models are resources for Rapid Contextual Design [42]), fusing analysis and data collection activities for quicker work.

The workshop and chapters contain some very good examples of appropriation and adaptation. The largest group of post-its at the workshop (91 or 37 %) was "Methods and Practices" ("Artefacts and Tools" comprised 16 of these). This identifies activity level resourcing as a key focus for Agile UCD. Roles and responsibilities are enabled or obstructed by the quality of their access to resources. However, expectations or preferences can result in ineffective use of resources. The desirability of formal prescriptive methods was discussed at the workshop, despite the common understanding (e.g., Chap. 10) that informal methods work best for agile [40]. Even formal methods cannot be 'followed' to the letter [41].

New resources are becoming available for Agile use. Chapter 2 provides evidence of the Internet of Things providing new resources in support of UCD activities, with SMEs tracking data from their software and hardware products. This complements web-originated customer and user information. However, Chap. 2 notes that good ideas remain hard to come by, so judgement in design management remains irreplaceable. A PO in one of the chapter's case studies makes use of simple heuristics based on frequency. A single suggestion or report of a problem is not enough, but 2–3 may be, but it may be that he: "stay[s] with [his] opinion". The extent of digital data and information here is still often filtered by a "feeling for", as well as by the road map for products and services.

Interaction Design support and tools are becoming more capable. Support is needed for keeping up to date with platform style guidance (Chap. 6) and rapid prototyping tools. The production quality and functional capabilities of these tools is increasing, and they can deliver code for development (e.g., HTML, CSS – Chap. 9). This makes it possible for UX work to 'keep up' with the pace of an agile project. High fidelity 'just in time' prototypes are now possible, and quicker to create than by drawing. Chap. 9 recommends progression from a clickable prototype, via a shell application with a partially functioning or no back end, to a fully functioning version released and in use: "it is ... actual usage that really validates the viability of the system."

User research and evaluation are the core contributions of UCD to software development. UX evaluation is often regarded as poor value for the time and resources expended. Agile can change this, and is doing so, by early release of

software for evaluation in use. This may be initially a restricted release [10], for example to living labs in Chaps. 2 and 6, or for release approval by a customer committee (Chap. 5). Once in full use, evaluation of digital products and services can draw on user support teams and marketing and business resources such as Net Promoter Scores (Chap. 2).

In summary, all development tasks are completed within or alongside iterations, which in turn form parts of an overall software development process. Support for tasks depends on resources, including the scheduling resources associated with sprints and processes. Ultimately, an extensive effective integration of Agile and UCD depends on the resources and approaches needed to support it. There are many examples of effective innovations here in this book's chapters, and many opportunities for further work, which are discussed in Sect. 1.3.

1.2.4 Research Methodologies

The topics listed in the call for participation for the NordiCHI workshop from which this book originates included theories and methods relevant to research on Agile and UCD. There was limited discussion of research methods at the workshop, where only three post-its related to research (within a larger group of" "research and problems", Chap. 8). However, collectively, this book's chapters make use of a wide range of research methods, which include:

- Literature surveys
- Theoretical analyses
- Surveys
- Interviews
- Observation, including participant observation
- Material culture studies (e.g., collection and analysis of Agile UCD arefacts such as user stories or bug/usability reports)
- Autobiographical reflection
- Grounded Theory, Meaning Consolidation, and other qualitative analyses
- Principal Components Analysis.
- Building mid-range theories, constructs and propositions from case studies
- Case studies
- Collaborative reflection
- Action Research (and Research through Design as a specific form of this)

Some of the above research methods, particularly case studies, are appropriate for early exploration of human practices that are still not well understood. Case studies can involve multiple methods: method mixes have been applied across multi-stage research projects and programmes lasting from 2 to 4 years in case study chapters.

There is no immediate clear pattern of groups across these methods. However, just as Kuusninen's distinction between teams and tasks (Chap. 9) provided a structure for the two previous sections, we can draw on a framework from

design research to support analysis of research approaches in this book's chapters. Although, the research methods above are largely associated by chapter authors with human science and action research practices, they way that they have been applied and combined is almost identical to well established design research practices.

Over 20 years ago, Frayling transferred a distinction between education *into* art and education *through* art to art and design research [43]. The former is the study of art, learning about its history and contemporary institutions. The latter is learning through the practice of art. Frayling thus contrasted *research into art and design*, with its historical and contemporary studies using humanities and human science methodologies, with *research through art and design*, with creative practices as the backbone of research methodology. Frayling also added a third mode of research in art and design, research *for* art and design. This mode developed knowledge and practices for use in art and design, e.g., knowledge of materials and design methods. Frayling also distinguished between 'Research' and 'research', i.e. original and significant academic *Research* in contrast to routine *research* within a professional practice (e.g., traffic surveys, epidemiology, food testing, market research). Focusing on art, Frayling found it difficult to come up with examples of *Research* for art, but found ready examples of artists' preparatory studies. He nevertheless referred to examples of *Research* for design (knowledge of materials and design methods), but failed to recognise these as original and significant academic research that could support future design practice and research.

The chapters in this book can be contrasted on the basis of Frayling's three modes of art and design research:

- *into*: studies of the agile development practices of *others*
- *through*: case studies based on researchers' *own* agile development practices
- *for*: research that results in guidance and practices for future design practice and research, e.g., in the form of knowledge, procedures, practices, principles or tools

In a study of design research PhDs, Yee concluded that Frayling's research modes are not mutually exclusive, but typically combine [44]. For example, chapters in the book could be classified as:

- research *into* design *for* design: studies of the agile development practices of others from which guidance and practices for future design practice and research are derived (e.g., Chap. 2)
- research *for* design *into* and *through* design: case studies based on secondary literature and researchers' own agile development practices from which guidance and practices for future design practice and research are proposed (e.g., Chap. 4)
- research *into* design *through* design *for* design: case studies based on researchers' own agile development practices and secondary literature from which guidance and practices for future design practice and research are derived from data and materials collected during a case study (e.g., Chap. 5)

Chapter 2's approach is *research into design for design*, i.e., existing SME practices are studied as a basis for offering suggestions for better integration of UCD and Agile. Stickel and colleagues positioned UCD as an evolving normative ideological

practice, with a primary focus on users, similar to Agile's focus on customers. However, their studies of Agile UCD 'in the wild', focused on what does happen rather than what should happen. Disparities between theory and practice in the literature make it important to study actual practice. They thus prefer a realist position over a normative one, and so draw on extensive qualitative research expertise to apply well established methodologies from design research to software contexts. The predominant research into design research mode here is essential for well-grounded understandings of when UCD and Agile can or cannot be successfully integrated. Differences exposed via research into design can explain disparities in the literature. For example, the two smaller SMEs studied in Chap. 2 are effectively part of larger virtual organisations in integrated value chains. Analysis here must consider the culture of the SME in focus, the cultures of their customers and partners, and the emergent dynamic culture of the embracing virtual organization.

Chapter 3's approach is *research for design through design*, i.e., it is research with the aim of developing support for design practice through iterative development of training materials and courses. New forms of support for Agile UCD were developed through practice-led iterations of curriculum and template design, with assessment through pre- and post-tests (after training, after independent use). Overall, Øvad and Larsen report the results of sustained critical sensitive informed and imaginative responses to developers' training experiences. The authors sensibly see their training suite as only one form of intervention in support of Agile UCD. Even so, it offers a highly effective strategy for addressing some outstanding challenges for Agile UCD.

Chapter 4's approach is *research for design through and into design*, where prototype development ('through design') is used to gain experience in Agile UCD, supplemented by a study ('into design') of agile practices in other teams in the same company, where team colleagues often worked on more than one project. This let the authors investigate the agile UCD practices of three other teams: two were working on the prototype development and other projects; the other was a UX team servicing several projects. Chapter 4 illustrates the advantages of experimental creative research through design practices. Sprint lengths were varied, 'invisible' UCD activities spanning sprints were actively supported, and Scrum Board configurations were iterated to adapt a well-established Agile resource to communicate the project's 'big picture' (in combination with two walls of a project space). These experimental creative research through design practices were *more agile than Agile*, in that they could change the rules in Scrum rather then be constrained by them. The results of these experiments were combined with the studies of three other teams to support research *for* design i.e., the researchers used the experience of their and other teams to offer guidance on running Scrum for future UCD focused projects, e.g.:

- conduct initial field studies in the first sprint, and develop infrastructure in the second sprint alongside user evaluations of lo-fi prototypes
- run UCD work one sprint ahead from Sprint 1 onwards, combining further field studies, lo-fi prototyping and user evaluations

- adopt ‘design chunking’ of closely related product features for implementation together in one sprint to let design and development activities manage dependencies. Design chunking addresses a known major issue with Scrum when dependencies between features are poorly managed. Costly refactoring of code can be required to correctly manage feature interactions
- do not have part-time staff on projects

Chapter 5 combined *research into and through design for design*. The initial research *into* design used secondary sources *for* co-designing an Agile HCD methodology that was suitable for the case study company and project. Positive experiences and best practices reported in this literature meant that Agile was not wholly negatively framed as obstructing HCD. An action research methodology supported the research *through* design in two ways. Firstly, the design of the web portal advanced the action research (above: creative practices as the backbone of research methodology). Secondly, the co-designed Agile HCD methodology also structured the action research process. In short, both the *product* and *process* of design were the means through which action research was carried out.

Chapter 6’s approach is research *into* design *for* design, i.e., Smart Campus practices provided experiences that were studied retrospectively (*into*) to advocate improving an organisation’s Agile UCD capabilities through PD and Design Thinking approaches (*for*). To understand the difficulties encountered in preserving PD practices, and how to overcome them, two interview studies were planned: the first occurred several months after the adoption of Scrum, the second one after over one year.

The imposition of Agile at a specific point in the project prevented early planning for integration (as was achieved by co-design in Chap. 5), which would have allowed a research through design methodology. Instead, the authors began with research into design, where they complemented their own project experience with interviews, which supported collaborative reflection to reduce bias, along with a researcher external to the project conducting the interviews, which were then transcribed and analysed, controlling for coding bias. A literature review in conjunction with the interview studies revealed how UCD work practices had failed following changes in the work situation; the authors benefitted from two 2014 literature surveys that covered 76 [12] and 71 [1] papers, which highlight the topicality of the chapters in this book.

Chapter 6 thus combines case study analyses, a literature survey, and collaborative reflection with software professionals. Grounding, analysis, modelling and guideline derivation were meticulous and reflect the strong empirical basis of almost all chapters in this book. The difference between design research and practice is evidenced by a project retrospective that was more rigorous, detached and systematic than any routine agile sprint or project retrospective could be.

In each case study for Chaps. 4, 5 and 6, Scrum-like processes were being used for the first time. In Chap. 4, Muñoz and colleagues decided to experiment, as a UCD focused team, with a Scrum process. In Chap. 5, Ardito and colleagues co-designed a Scrum inspired process and then applied it in an action research project. In contrast in Chap. 6, Bordin and De Angeli found Scrum imposed on their research

project. We thus see differences in autonomy in these three practice-based case studies. Chapters 4 and 5 case studies were largely under the researchers' control. Chapter 2 presented three SME case studies without action research interventions. Chapter 3 reported on the progress and outcomes of training interventions with agile developers.

Chapter 7 also addresses the capabilities of agile developers by assessing the potential for developers taking on more UCD work, as in Chap. 3. This research *into* design study focused on the capabilities of specific roles in a project team, rather than on the broad context of agile development, as in Chap. 2's qualitative study, but Chap. 7 uses a quantitative method, principal component analysis of data from UX rating scales. This straightforward research *into* design *for* design investigates opportunities for using UX ratings in Agile UCD. It does not lead to clear guidance, since the results are somewhat surprising and the sample of specialist roles is very small in comparison to developers and users. Although UXS and PO roles were better than developers at predicting users' ratings of product UX for case studies, they were not good enough to recommend using professional's ratings (even when taking the users' perspective) as a substitute for empirical evaluation involving end-users. However, the study is valuable for two reasons. Firstly, it calls the empathy of UXS and PO roles into question. UXS are meant to be the users' advocate, and POs are meant to be the customer's advocate, but the small sample do not appear to be well enough equipped for this role (moreover, perfect empathy may well be impossible for anyone). Secondly, it suggests that users' UX ratings could provide useful support for iteration (sprint) planning. Further research is required to: establish whether the reported results will change with a larger sample of UXS and PO roles; and explore the value of users' UX ratings in Agile UCD.

Chapter 8 reports a short collaborative research *into* design workshop that provides a future research agenda for a range of mixes of research *into*, *through* and *for* design.

Chapter 9 uses a co-ordinated sequence of research *into* design studies to develop a research *for* design output: the BoB framework, ending with collaborative reflection to reduce bias. The BoB framework is based on four mixed method studies spanning 4 years and involving over 300 respondents from 9 companies in 10 countries (7 in Europe and 3 in Asia), working across IT services, engineering, middleware, mobile enterprise applications, and industrial systems (including safety critical).

Chapter 10 takes a similar research *into* design to suggest some novel tactics *for* improving the diffusion of UCD. Chapters 9 and 10 have both passed through the first stage of the research programme methodology proposed in [41]: detailed, well-structured case studies of usability work. They both reach the second stage: metareviews of case studies of usability work. Chapter 9 has reached the third stage: modeling the interaction design process across a complete project life cycle, based on the results of a meta-review. The stages of the proposed research programme structure in [41] are designed to support each other, with the aim of reaching a fourth stage where well-grounded and theorized conjectures about the impact of resources and approaches can be formulated and tested. The BoB framework makes

formulation and testing of such conjectures possible. Its supporting guidelines (Chap. 9 Table 9.3) can be read as a first outline of a coherent set of related conjectures.

Chapter 11 is a research *into* design study using secondary sources that moves to research *for* design by outlining how Agile UCD can expand its current scope via BIG (Balanced, Integrated and Generous) design [45]. Collectively, the case studies span much of the scope of BIG design, but no single case study explicitly set out to span this scope, nor did any cover the full scope in isolation.

Action research practices in case studies (research *through* design [43]) have proved effective and have created a range of resources that can be adopted and adapted for Agile UCD. These include training materials, workplace layouts, process structures, and sprint practices (research *for* design [43]). Development and assessment of new approaches in practice is a research methodology that is suitable for applied projects in universities and industry, where it can benefit from collaborative reflection (e.g., Chaps. 2 and 9). Frayling's modes of art and design research thus provide a useful framework for contrasting the high level research strategies taken in each of the book's chapters, which significantly improve on the workshop position papers in their coverage of theories and methods relevant to research on Agile and UCD. The framework provides a structure that can make sense of the diverse range of research methods applied in each chapter. Frayling's research modes also provide a basis for identifying future methodological work for Agile UCD, as well as framing best practice. For example, future action research methodologies for Agile UCD could combine the planning of Chap. 5 with the systematic iteration of resources in Chap. 3.

1.3 The Future of Agile UCD Practices and Research

There has been successful Agile UCD work for over a decade, which chapters in this book augment with new successes from ambitious resourceful and innovative software development teams. Nevertheless, the previous section has identified continuing challenges. Many challenges can be addressed via a combination of professional innovation and applied academic research. Some challenges however require more focused fundamental research, which could further develop methodologies at the leading edge of Agile, UCD and creative design research.

Two topics are addressed in this section: the continuing challenges for Agile UCD; and future trends for research on UCD and Agile.

1.3.1 Continuing Challenges for Agile UCD

Novel approaches and resources developed in this book's case studies will not transfer to all development contexts. Some will not scale to larger teams who are

not co-located, so the current solutions are not viable in all agile contexts. Chapter 6 also raises issues of transfer to Open Source contexts, which remain a challenge for Agile.

Domain values give rise to specific needs [46]. Some application domains are not addressed in this book, or example, no chapter addresses games. Challenges for Agile here include the need to completely implement a game's narrative for adequate gameplay. Similarly, at least a few game levels need to be in place for the first release of a game. This need for comprehensive capabilities at first release may reduce the benefits of following Agile in games and perhaps other domains such as healthcare, once systems significantly exceed the complexity of focused specialist products (such as those developed by a Chap. 3 SME).

There are thus continuing challenges of extending Agile UCD to further application domains, application complexity, team sizes and open source contexts. The case studies also indicate a need to develop better tool support to improve Agile UCD work. When widely available tools (as in Chaps. 2, 4, 5, 6 and 9) are used for new Agile UCD activities ('appropriation'), this will improve diffusion, as will use of (improved) open source tools.

Some continuing challenges are specific to organizational settings. A workshop post-it in response to Chap. 5's position paper: can an action-research-based approach be used to alter the 'call for tenders'? What is interesting here is the reframing of a longstanding UCD issue as an action research problem. However, experimental interventions here would be limited to IT service companies and their customers. For mass market product and services, this is a question for project inception. Either way, action research here must involve suppliers and customers.

Some continuing challenges are specific to UCD. Chapter 9 suggests using personas [47] to develop shared user empathy. These could be co-created by designers and developers, building on Chap. 3's preference for independent hands-on creation of project resources over observe and learn. Support for estimation beyond feature implementation is also required. In Chap. 4, the existing agile practice of Planning Poker did not transfer successfully to an experimental Agile UCD context. Similarly, iteration (sprint) planning must expand beyond a feature focus to reduce constraints on creativity and flexibility (Chap. 6), but this is a joint challenge for Agile and UCD. Such multi-functional iteration (sprint) planning will better align design and development, by improving support for novel approaches to sprint management, as in Chaps. 4, 5 and 9.

Some continuing challenges are more general and are all linked to Agile becoming more mature and mainstream. Scaling and sustainability are still concerns. A single methodological or process success on a small project that is intended for general use must translate into sustainable success on larger projects. Also, the whole development process needs to be covered. Chapter 5 shows awareness of life before upfront development activities (i.e., project inception), pointing to a need to research process planning from cradle to grave, and not just from kick-off to product launch. Strategic visions exist in some form before kick-off workshops, but these are not well integrated into either UCD or popular agile approaches. Chapter 9's BoB framework has an early product definition workshop, and Chap. 5 outlines

its lengthy Sprint n.0. Research needs to provide details and examples to better support Agile UCD from project inception onwards. For example, User Stories [48, 49] need to be linked to a holistic big picture that puts them in a strategic context. A big picture can also support Chap. 4's chunking of feature groups to support of UX quality [33, 34]. Similarly, an overall project vision and roadmap is vital for strategic filtering of user and customer feedback (Chap. 2), since a major problem for UCD is that users' needs aren't all the same, even for a largely coherent subgroup represented by a persona archetype [47]. Upfront activities are thus vital to creating and documenting this big picture, e.g., as a vision or roadmap.

Merging, interpreting and filtering user feedback was a challenge that was in focus in Chaps. 2 and 6. Given that customers want user feedback (and suppliers want customer feedback), users and customers must be motivated to provide feedback, and to keep doing so, especially in Participative Design contexts (PD: Chap. 6). Feedback on feedback is thus another Agile UCD practice that requires further applied research in practitioner and research contexts. To be able to provide feedback on feedback, feedback needs to be effectively collected and efficiently analysed, which is becoming a major challenge with the development of a wide range of crowdsourcing channels (e.g., web, social media, instrumentation, user support, living labs) alongside formal evaluation activities (which may be outsourced). An initial research agenda here includes investigating appropriate balances of quantitative and qualitative data, making use of existing CSCW and Mobile HCI research (e.g., [50, 51]) and associated tools (e.g., the open source Shake tool from Chap. 2's authors and colleagues: http://github.com/UniSiegenCSCW/Shake).

In summary, challenges remain with Agile UCD for some application domains:

- larger project teams;
- open source contexts;
- tendering;
- estimation;
- multi-functional iteration (sprint) planning;
- whole process planning;
- creating and maintaining a project vision and keeping development activities focused on it, and efficiently and effectively managing increasingly complex user feedback.

These continuing challenges can be largely addressed by action research within Agile UCD, with or without the support of academic researchers, using hybrid practice-based design research methodologies that combine Frayling's modes [43], as in this book's case studies. Chaps. 3, 4 and 5 provide inspiring examples of invention and reflection that can be emulated in further practice-based research on process planning and management, on multi-channel evaluation data, and on documentation. Future research in these areas needs to report: tactics (e.g., developer training); example novel and adapted resources and approaches (e.g., extranet Scrum Boards, triangulation of customer/user feedback, kick-off workshops, design sprints [28], pair working); and systematic research reports on Agile UCD experience.

Careful record keeping and critical collaborative reflection are needed to ensure high research quality. Rigorous reflective experience reports need to be able to identify the separate contributions of team composition and work structure, and how these combine in practice. Support and impedance from organizational cultures needs to be identified, along with the role of Agile, UCD, developer, designer, customer and user values. It is important to contextualize experience reports so that readers can assess how they match their own development contexts. Alongside this, following Chap. 3, experimental studies also need to aim to develop agile research for agile practice (and following Chaps. 4, 6 and 11, to develop creative research for creative practice).

1.3.2 Future Trends for Research on Agile and UCD

Some challenges for Agile and UCD, both integrated and in isolation, require sustained research interventions across multiple projects with the involvement of academic researchers, professional communities and key customers. Successful research here will require research methodologies with sophistication beyond those used in this book's chapters. The case study methodologies in this book do not lack sophistication, but the research challenges for some future work are substantial.

Practices are always changing and developing, so continued empirical research is essential: good ideas can come from academics and practitioners working together. Variety in research methods gives different insights, academic rigour is helpful, and academics are prepared to look at failures as well as successes (both are limitations of the many experience reports in the agile literature). Theoretical views also help to move understanding forward, and reframe discussions. However, pragmatic outputs such as models, templates, process descriptions and techniques are helpful for practitioners. Most importantly, researchers need to get better at getting their findings and insights back out to the practitioner community so that they can learn from and use them. The latter applies to all of the recommendations on future research below, which are not exhaustive. There is no 'definition of DONE' for Agile UCD research. There will always be new challenges and new issues to explore, but maybe one of the most important challenges is to maintain collaboration between researchers and practitioners. With these pervasive challenges in mind, more challenging research areas include:

1. the focus and quality of surveys on Agile, and their ability to accurately represent trends and actual practices
2. length of experience with Agile and openness to integrating UCD and other practices such as creative design and innovation processes
3. longitudinal studies of customer value and UX from Agile UCD
4. managing a broad range of non-functional requirements beyond UX
5. adapting Agile UCD to novel technologies and application domains

The 'Scrumish' nature of practices across all the case studies suggest that the results of key surveys such as VersionOne's [29] must be interpreted in the context

of what is most probably very loose self-reporting. Two respondents reporting themselves as Scrum users may well have very different agile work practices. Knowing what is actually happening at an appropriate level of detail will not be revealed by large quantitative surveys. Qualitative literature surveys [1, 12] are currently better at revealing the detail and nuances of Agile UCD practices. However, follow up interviews with an appropriate sample of respondents to large online questionnaires could well improve on current literature surveys that are likely to be a few years behind current practices.

Balancing quantitative surveys with qualitative interviews can better answer question about what is actually happening in Agile, beyond self-reporting of methodology names. A question raised by one position paper [7] was: is Scrum going out of fashion and being replaced by Lean methodologies such as Kanban? This is not an easy question to answer. However, what is happening is that organisations are creating their own Agile mixes, which may lower barriers to UCD. The case studies all use local Agile mixes rather than one specific exclusive 'branded' methodology. However, there is a sense for some researchers currently that elements of Lean or Kanban are making inroads into Scrum practices, but respondents to large surveys may still report that they are just using Scrum. If Lean or Kanban are becoming more common, in whole or in part, then future research needs to focus on integrating UCD here as well as with Scrum.

As Sect. 1.2.1.2 above (Values other than Agile and UCD) has noted, understanding 'what is happening in Agile UCD' involves more than just understanding contemporary Agile and UCD. What appears in this book's case studies is not simply openness to Agile or UCD, in fact it is much simpler, i.e., simply openness. There is evidence of openness to approaches from creative design, innovation, business and SE (software engineering), with examples of each integrated into agile practice.

A further drawback on annual quantitative surveys is that they only provide a snapshot in time, as do qualitative case studies. However, given that method mixes constantly evolve, a question follows as to whether there are points in an organisation's experience with Agile or UCD that are more favourable to integrating other practices. In the case studies in this book, novice use of one approach is supported by experienced use of another. Thus experienced UCD teams' first use of Scrum with UCD, with local adaptations, was generally successful in the Chaps. 4, 5 and 6 case studies (although Chap. 6 reports frustrations related to Participative Design). Conversely, in Chaps. 2, 3, 7 and 9, organisations with years of Agile experience successfully integrated UCD practices. There is thus no broad evidence that commitments to Agile completely obstruct UCD practices. UX roles are not in charge, and may feel excluded from sprint and similar meetings, but the issue here is one of degree, i.e., how, when and to what extent ideal UCD practices are not possible. The question of maturity and readiness is thus not straightforward, and asking it may currently be premature. Instead, exploratory longitudinal studies are required to track the evolution of method mixes in organisations that initially primarily use Agile or UCD, and those that start with both or neither at similar stages of development.

Organisations need to combine perspectives and approaches from SE, UCD and business to deliver the customer value that Agile aims at. Research is needed to establish how UX and customer value do or do not develop post release. This also requires sophisticated longitudinal research methodologies. At organizational level, consumer market SMEs in Chap. 2 already have practices to track this. For example, one PO shares his phone number with experts in their user community. However, research needs to look for patterns across larger samples of organisations.

SE has always been challenged by non-functional requirements [2] such as UX and customer value. A research question on the development of UX and customer value can be extended to more non-functional requirements, e.g., security, sustainability, maintainability, privacy and other ethical concerns. Chapter 5 described how non-functional requirements for internal software quality were monitored in its case study. Chapter 10 reports how external software qualities [2] such as security and accessibility are used as 'trojan horses' to sneak in usability requirements. UCD has often taken a moral position, but Chap. 10 shows that this is not as fashionable as Agile's values.

As smart, transmediated and pervasive technologies support workplace users and become a key to efficient organisations, ethical issues such as work-life balance become more important. Emerging digital technologies and the complex landscape of the many application domains where they can be used make UX work more complex, and increase the challenges of conducting high quality research in these areas. Chaps. 2, 3, 6 and 9 cover extensions of Agile UCD to mobile technologies and hardware based ('embedded') systems. In technical terms, Agile has extended into new application domains using novel technologies, but UCD may not be keeping up. The training regime in Chap. 3 can help companies catch up here, but new approaches are needed for non-functional requirements in contexts for new applications of novel technologies.

This concludes discussion of elements of a future collaborative practice-based research agenda. Other research needs to be more academic in nature, as it relates to one workshop call focus: theories and methods relevant to research on Agile and UCD. This section closes with a discussion of each.

One of the current limitations of the Agile literature is that much of it consists of 'experience reports' where rigour is typically weak. There is also a tendency only to publish 'success stories' rather than failures. Research approaches such as single and multiple case studies, action research, and ethnography should improve rigour, especially when these are integrated through adoption and adaptation of methodologies from research through design (also known as constructive design research [52]).

Documentation is key to rigorous research through design, which offers an experimental setting that can be used to develop practices that can be transferred to practitioner contexts. Documentation, preferably created, maintained and accessed via shared extranet tools, underpins the underdeveloped areas of Agile UCD, i.e., process planning, aligning design and development, feedback management, and communication within project teams. Experimental research needs to develop tactics, resources and approaches. Mixed media documentation practices (Chaps. 4

and 6) are required that can curate code, sketches, architectures, prototypes, text, data, and images.

Future methodological directions for foundational Agile UCD research can thus benefit from developments in design research that combine appropriate documentation, critical reflection and creative practice for rigour in open ended experimental studies. This is primarily an area for academic research until methodological issues are resolved, particularly in relation to appropriate documentation and critical reflection.

Explicitly framing of Agile UCD research projects as hybrid forms of design research will guide researchers to draw on relevant theories, results and methodologies from design research. For example, a range of research through design methodologies is covered in [52], and HCI and design research conferences regularly update this with new work on practice-based methodologies.

Existing UCD research can also contribute to improving Agile UCD research. Chapter 2 describes how one PO used a heuristic similar to any two agreement [53], but this is known to risk missing severe problems [54]. Similarly, Chap. 2 argues that crowdsourcing approaches to evaluation can exploit "concepts from HCI and CSCW, such as comprehensive situated user feedback and engagement mechanisms right inside of products, or leveraging modern mobile devices to facilitate relatively lean, event-contingent qualitative and quantitative data collection".

Another advantage of academic research in this area is that we can use theory to help to understand and hence improve practice. Theories relevant to research on Agile UCD in the chapters are largely drawn from the business and innovation literature. Chaps. 6 and 11 draw on Design Thinking (as did [6]). Chapter 10 draws on Management Fashionability and Diffusion of Innovation. Chapter 11 draws on design research, which underpins much innovation work [13, 19, 20, 22, 27, 45]. These are not mainstream theories in HCI or SE. They evidence a shift from seeing software development as solely a concern for Agile and UCD, and extend the scope of challenges to a broader range of disciplinary practices. Theories from Organisational Science, Psychology, Sociology and Communications Science are also likely to be relevant to improving our understanding of Agile UCD in its broader organizational, professional and business contexts. Also, theories from design research are relevant.

A simple question has been asked since the dawn of UCD, and was asked at the workshop in relation to a position paper [7]: "What is good and bad in different methods?" This question is simple to ask, but very difficult to answer. Even harder to address is a need expressed at the workshop following the position paper for Chap. 3: a method for validating UCD methods, including their adapted forms. Given that the methods communicated in the training materials were revised for each iteration, this in some ways was a surprising question. The position paper celebrated agile research for agile practice, and a requirement for rigorous validation would significantly reduce agility here. Even so, the pre- and post-tests methodology applied in Chap. 3 did show improvements with each iteration of each method, but this was largely in terms of developers' perceptions and not whether each version of each method delivered 'correct' results when it was applied.

A need to validate and evaluate methods reflects engineering and human science values. However, there is a fundamental error in regarding methods as independent variables that have fixed effects in highly variable contexts. Less demanding assessments of method innovation and adaptation in Agile UCD can however be on the research agenda. Chaps. 2, 3 and 6 use qualitative research methods that reveal practices, but without attempting to assess their validity for every development context. There is scope for methodological improvement here, but an important body of theory in design research [41, 45] strongly advises against any attempt to 'prove' that 'methods' are valid. This is because 'methods' are an inappropriate unit of analysis for design and evaluation work. It is possible to establish reliable impacts for resources, at a lower level of analysis, and project level factors at a higher level. However, methods are too underspecified for there to be reliable 'method effects'. Methods are the result of development work, and not inputs to them [41]. The nature of creative work is such that methods cannot be systematically followed. Chapter 11 presents the underlying theory here.

Future research on Agile UCD can apply a framework from design research [45] to research *into* design studies such as Chap. 2's case studies. The framework in [45] replaces the resource *types* in [41] with resource *functions* that can identify work done at specific points in activities. For example, a resource can be informative or inquisitive, or it may direct work (e.g., Chap. 3's templates). Resource function analysis was used in [55] to identify the reasons for the success of an unplanned evaluation approach, Creative Sprints. The hybrid design research approach here was thus research *into* design *for* design. It identified organisational requirements for effective application of Creative Sprints.

Resource function analysis can also be used to support collaborative reflection. Chaps. 6 and 9 benefitted methodologically from collaborative reflection, and resource function analysis guided collaborative reflection in [55]. The results of resource analyses such as [41] can inform training approaches (as in Chap. 3) to improve evaluation performance in Agile UCD via relevant research.

There are thus many opportunities for improving the methodological and theoretical support for Agile UCD research.

1.4 Conclusions

The chapters developed after the NordiCHI 2014 workshop make substantial contributions in relation to all topics in the workshop's call for participation:

- Case studies and work in progress related to UX and Agile.
- Success stories and best practices from integrating UCD and Agile.
- Challenges from working with UCD in Agile systems development
- Values and perspectives underpinning UCD and Agile in theory.
- Integration of UCD and Agile in different domains such as games and healthcare.
- Theories and methods relevant to research on Agile and UCD.
- Discussion of future trends for UCD and Agile research.

In conclusion, some key observations can be offered. Firstly, Agile UCD requires give and take. Both Agile and UCD can learn from each other, and from other approaches from design, engineering and business. Chapter 2 argues that Agile provides better support for work-based learning than UCD does. UCD's origins in the human sciences tend to favour library-based learning.

Secondly, UCD can be difficult, but so can SE in agile and other guises. Agile UCD by extension is even more difficult. In such circumstances, the good natured collaboration between UCD and SE professionals in most of the case studies (most of the time) is at least as important to research success as methodological rigour.

Thirdly, Agile UCD is going to get even more difficult as it becomes better interfaced with design thinking and business strategy. The big picture will get bigger. Progress on tactics will always be needed, but strategic capability will become the dominant goal. Interactive systems are no longer just supporting workers on tasks. They now underpin enterprises and services. In some senses, IT systems are becoming *the* business, rather than just supporting it. Narrow foci will not be sustainable, whether these foci are on UX or internal software quality: both respectively represent legitimate UCD and SE concerns that have to co-exist with, and take their direction and goals from, the wider enterprise contexts within which contemporary IT systems are developed. Both Agile and UCD will have to face both inward and outward: inward towards their professional and disciplinary concerns, and outward to the value and experiences that they aim to deliver. Agile UCD is only one step in a larger process of integrating software development and business functions in support of the digital economy and digital social and cultural innovation.

References

1. Salah D, Paige RF, Cairns P (2014) 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, vol 5. ACM, New York
2. Gram C, Cockton G (1996) Design principles for interactive software. Chapman and Hall, London
3. Boehm B, Turner R (2003) Balancing agility and discipline: a guide for the perplexed. Addison-Wesley, Boston
4. Meyer B (2014) Agile! The good, the hype and the ugly. Springer, Switzerland
5. Lárusdóttir M, Cajander Å, Gulliksen J, Cockton G, Gregory P, Salah D (2014) On the integration of user centred design in agile development. In: Proceedings of the NordiCHI '14. ACM, New York, pp 817–882
6. Lindell R (2014) Attending experiential qualities in system development. In: Proceedings of the workshop: on the integration of UCD and agile development, NordiCHI 2014, Helsinki. Available from https://ucdandagile.wordpress.com
7. Law ELC, Lárusdóttir MK (2014) User Experience (UX) Design: agile or lean?. In: Proceedings of the workshop: on the integration of UCD and agile development, NordiCHI 2014, Helsinki. Available from https://ucdandagile.wordpress.com
8. Law EL-C, Lárusdóttir MK (2015) Whose experience do we care about? Analysis of the fitness of scrum and kanban to user experience. Int J Hum Comput Interact 31(9):584–602

9. Nelson E (2002) The Beck-Cooper interview: extreme programming vs. interaction design. Available from http://www.id-book.com/downloads/beck_vs_cooper_debate.pdf
10. Ambler SW (2008) Tailoring usability into agile software development projects. In: Law E, Hvannberg E, Cockton G (eds) Maturing usability. Quality in software, interaction and value. Springer, London
11. Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W and 12 other authors (2001) Manifesto for Agile Software Development. http://www.agilemanifesto.org/
12. Jurca G, Hellmann TD, Maurer F (2014) Integrating Agile and user-centered design: a systematic mapping and review of evaluation and validation studies of Agile-UX. In: Agile Conference (AGILE), 2014, IEEE (2014), 24–32
13. von Hippel E (2005) Democratizing innovation. MIT Press, Cambridge, MA
14. Keiningham TL, Cooil B, Andreassen TW, Aksoy L (2007) A longitudinal examination of net promoter and firm revenue growth. J Mark 71(3):39–51
15. Basili VR, Caldiera G, Rombach HD (2002) Experience factory. In: Encyclopedia of software engineering. Wiley, New York
16. Schneider K, von Hunnius J-P, Basili V (2002) Experience in implementing a learning software organization. IEEE Softw 19(3):46–49
17. Gartner S, Schneider K (2012) A method for prioritizing end-user feedback for requirements engineering. In: 5th International workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE 47–49
18. Lee MJ, Ko AJ (2012) Representations of user feedback in an Agile, collocated software team. In: 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE 76–82
19. Brown T (2005) Strategy by design, fast company, June 2005. http://www.fastcompany.com/52795/strategy-design
20. Brown T (2009) Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation. HarperBusiness, New York
21. Gould J, Lewis C (1985) Designing for usability: key principles and what designers think. CACM 28(3):300–311
22. Lavie T, Tractinsky N (2004) Assessing dimensions of perceived visual aesthetics of web sites. Int J Hum Comput Stud 60(3):269–298
23. Wang K (2015) The ultimate guide to hamburger menus and their alternatives. http://apptimize.com/blog/2015/07/the-ultimate-guide-to-hamburger-menus-and-alternatives/
24. Sutherland J (2003) SCRUM: get your requirements straight before coding. https://www.scruminc.com/scrum-get-your-requirements-straight/
25. Cockburn A, Highsmith J (2001) Agile software development: the people factor. IEEE Comput 34(11):131–133
26. Highsmith J, Cockburn A (2001) Agile software development: the business of innovation. IEEE Comput 34(9):120–127
27. Ungar JM, White JA (2008) Agile user centered design: enter the design studio – A case study. In: Proceedings of the CHI 2008. ACM, New York, pp 2167–2177
28. Knapp J (2016) Sprint: how to solve big problems and test new ideas in just five days. Simon and Schuster, New York
29. VersionOne (2015) State of Agile™ Survey. www.versionone.com/pdf/state-of-agile- development-survey-ninth.pdf
30. Porter S (2011) Chickens and pigs, scrum.org community publications Article 90, https://www.scrum.org/About/All-Articles/articleType/ArticleView/articleId/90/Chickens-and-Pigs
31. Plonka L, Sharp H, Gregory P, Taylor K (2014) UX design in agile: a DSDM case study. In: Agile processes in software engineering and extreme programming: 15th International conference, XP 2014, Lecture Notes in Business information processing, Springer
32. Cockton G (2008) Revisiting usability's three key principles. In: Czerwinski M, Lund AM, Tan DS (eds) CHI 2008 extended abstracts 2473–2484
33. Sy D (2007) Adapting usability investigations for agile user-centered design. J Usability Stud 2(3):112–132

34. Sy D, Miller L (2008) Optimizing Agile user-centred design. In: CHI '08 extended abstracts on human factors in computing systems (CHI EA 08), ACM, 3897–3900
35. Cockton G (2012) UCD: critique via parody and a sequel. In: Proceedings of the CHI 2012 extended abstracts on human factors in computing systems (CHI EA '12). ACM 1–10
36. DSDM Consortium (2014) The DSDM Agile project framework. DSDM Consortium Ebook: https://www.dsdm.org/resources/dsdm-handbooks/the-dsdm-agile-project-framework-2014-onwards
37. Shuja AK, Krebs J (2007) IBM Rational unified process reference and certification guide: solution designer (RUP). Pearson Education, Indianapolis, IBM Press
38. Brhel M, Meth H, Maedche A, Werder C (2015) Exploring principles of user-centered agile software development: a literature review. Inf Softw Technol 61:163–181
39. da Silva T, Martin A, Maurer F, Silveira M (2011) User-centered design and Agile methods: a systematic review. In: Proceedings of the agile methods in software development (Agile 2011)
40. Jia Y (2012) Survey on scrum and UCD. Master thesis, Uppsala University
41. Woolrych A, Hornbæk K, Frøkjær E, Cockton G (2011) Ingredients and meals rather than recipes: a proposal for research that does not treat usability evaluation methods as indivisible wholes. Int J HCI 27(10):940–970
42. Holtzblatt K, Wendell JB, Wood S (2005) Rapid contextual design: A how-to guide to key techniques for user centered design. Elsevier, San Francisco
43. Frayling C (1993) Research in art and design. RCA Res Pap 1(1)
44. Yee JSR (2010) methodological innovation in practice-based design doctorates. J Res Pract 6(2), Article M15. http://jrp.icaap.org/index.php/jrp/article/view/196/193
45. Cockton G (2013) Design isn't a shape and it hasn't got a centre: thinking BIG about post-centric interaction design. In: Proceedings. MIDI '13. ACM, Article 2, p 16
46. Cockton G (2015) Domain Values and Method Transferability: an Initial Framework. In: Christou G, Zaphiris P, Law ELC (eds) 1st European workshop on HCI design and evaluation, IRIT Press, p 85–90. www.irit.fr/recherches/ICS/projects/twintide/upload/427.pdf
47. Adlin T (2010) The essential persona lifecycle: your guide to building and using. Morgan Kaufmann, San Francisco
48. Cohn M (2004) User stories applied. Addison-Wesley Professional, Boston
49. Kaczor K (2011) 5 common mistakes we make writing user stories. Scrum Alliance Member Article. https://www.scrumalliance.org/community/articles/2011/august/5-common-mistakes-we-make-writing-user-stories
50. Yetim F, Draxler S, Stevens G, Wulf V (2012) Fostering continuous user participation by embedding a communication support tool in user interfaces. AIS Trans Hum Comput Interact 4(2):153–168
51. Dax J, Ludwig T, Meurer J, Pipek V, Stein M, Stevens G (2015) FRAMES – a framework for adaptable mobile event-contingent self-report studies. In: Diaz P, Pipek V, Ardito C, Jensen C, Aedo I, Boden A (eds) End-user development, LNCS, 9083. Springer, Dordrecht, pp 141–155
52. Koskinen IK, Zimmerman J, Binder T, Redström J, Wensveen S (2011) Design research through practice: from the lab, field, and showroom. Morgan Kaufmann, Waltham
53. Hertzum M, Molich R, Jacobsen NE (2014) What you get is what you see: revisiting the evaluator effect in usability tests. Behav Inform Technol 33(2):143–161
54. Woolrych A, Cockton G (2001) Why and when five test users aren't enough. In: Vanderdonckt J, Blandford A, Derycke A (eds) Proceedings of the IHM-HCI 2001, Joint AFIHM-BCS Conference on human-computer interaction vol II, Cépadèus Éditions
55. Garnik I, Sikorski M, Cockton G (2014) Creative sprints: an unplanned broad agile evaluation and redesign process. In: Proceedings of the NordiCHI '14. ACM 1125–1130