In: Ali R., Kaindl H., Maciaszek L.A. (eds) Evaluation of Novel Approaches to Software Engineering.
ENASE 2020. Communications in Computer and Information Science, vol 1375. Springer

HumaniSE: Approaches to Achieve More Human-Centric Software Engineering

John Grundy[0000−0003−4928−7076], Hourieh Khalajzadeh[0000−0001−9958−0102], Jennifer McIntosh[0000−0002−6655−0940], Tanjila Kanij[0000−0002−5293−1718], and Ingo Mueller[0000−0003−2240−712X]

HumaniSE Lab, Faculty of IT, Monash University
Clayton VIC 3800, Australia
{John.Grundy, Hourieh.Khalajzadeh,
Jenny.McIntosh,Tanjila.Kanij,Ingo.Mueller}@monash.edu
https://www.monash.edu/it/humanise-lab

Abstract. A common problem with many existing software systems and the approaches to engineering them is their lack of the *human aspects* of their target end users. People are different - with diverse characteristics including age, gender, ethnicity, physical and mental challenges, personality, technical proficiency, emotional reactions to software systems, socio-economic status, educational attainment, language, and so on. In this paper we describe our work at looking to better consider these characteristics by incorporation of *human aspects* throughout the software engineering lifecycle. We are developing a co-creational *living lab* approach to better collect human aspects in the software requirements. We are using *domain-specific visual languages*, themselves a more human-centric modelling approach, to capture these diverse human aspects of target software systems. We are working on incorporating these human aspects into design models to support improved *model-driven engineering*, and thereby to better support both code generation and run-time adaptation to different end user human characteristics. Finally we are working on better ways to support *continuous evaluation* of human aspects in the produced software, and to provide improved feedback of user reported defects to developers.

Keywords: Model-driven engineering · Human-centric software engineering · human factors.

1 INTRODUCTION

Modern software systems are extremely complex, currently hand-crafted artefacts, which leads them to be extremely brittle and error prone in practice. We continually hear about issues with security and data breaches (due to poorly captured and implemented policies and enforcement); massive cost over-runs and project slippage (due to poor estimation and badly captured software requirements); hard-to-deploy, hard-to-maintain, slow, clunky and even dangerous

solutions (due to incorrect technology choice, usage or deployment); and hard-to-use software that does not meet the users' needs and causing frustration (due to poor understanding of user needs and poor design) [7, 47, 62]. This leads to huge economic cost, inefficiencies, not fit-for-purpose solutions, and dangerous and potentially even life-threatening situations. Software is designed and built primarily to solve human needs. Many of these problems can be traced to a lack of understanding and incorporation of these *human aspects* during the software engineering process [23, 41, 58, 60, 54]. This includes aspects such as age, gender, language, culture, emotions, personality, education, physical and mental challenges, and so on.

Current software engineering approaches ignore many of these human aspects, or address them in piece-meal, ad-hoc ways [47, 7, 23, 60, 4]. For example, in Model-driven Software Engineering (MDSE), user requirements for the software are captured and represented by a variety of abstract requirements models. These are then refined to detailed design models to describe the software solution. These design models are then transformed by a set of generators into software code to implement the target system [52]. However, currently almost no human aspects are captured, reasoned about, designed in or used when generating or testing the software produced in this way [41, 62]. We need to more fully integrate these *human aspects* into model-driven software development.

This paper is an extended version of an earlier one that appeared at ENASE 2020 [19]. In this paper we describe in more detail, by using a number of example projects, how we are addressing these issues using several complementary approaches, as outlined below:

Use a co-creational, agile Living Lab-based approach: our idea here is to better enable software teams to provide better ways for software engineers to work with stakeholders to capture and reason about under-represented, under-used, under-supported yet critical human-centric requirements of target software. Our Living Lab is designed to provide a co-creational space for investigating socio-technological aspects of software engineering activities. Our focus is on enabling software development teams to capture and reason about critical human-centric aspects of software.

Develop a new set of human-centric requirements modelling languages: Understanding software stakeholder needs is essential for a successful software development project. However, software developers tend to focus much more on technological aspects and therefore often do not sufficiently capture the complete human context concerning, for example, software user age, accessibility challenges, ethnicity, language or gender. We want to enable software engineers to more effectively elicit and model diverse human aspects. Along with this we need new approaches for obtaining and extracting such human-centric software requirements from a wide variety of sources e.g. Word, PDF, natural language, videos, sketches, and so on.

Augment conventional model-driven engineering design models with human-centric requirements: we want to be able to use human-centric aspects during MDSE, along with techniques to verify the completeness, cor-

rectness and consistency of these models, and proactively check them against best practice models and principles. We believe that incorporating more human-centric aspects into software design will lead to more useful, usable and desirable software. We work towards this goal by creating new notations, techniques and tools to augment software design models with human-centric aspects. We also need to develop new techniques to incorporate these human aspects in design models into MDSE-based software code generators, enabling target software to dynamically adapt to differing user needs at run-time.

Use of these human-centric requirements during software testing: Software testing and evaluation is an essential software development activity which is aimed at ensuring that requirements have been implemented correctly and completely and that user needs are met. We focus on the testing of human-centric aspects and techniques for receiving better user feedback on human-centric defects in their software solutions. We need to better support human-centric requirements-based testing of software systems, along with techniques that give developers better feedback from users on the human aspects-related defects in their software solutions.

The rest of this paper is organised as follows. Section 2 presents a motivating example for this work, along with a review of key related work. Section 3 provides an overview of our approach. Section 4 discusses our development of a co-creational living lab approach, and Section 5 the use of human aspects during requirements engineering. Section 6 presents some projects we are undertaking enhancing design and model-driven software engineering to incorporate human aspects of end users. Section 7 discusses approaches to evaluating software as to how well (or poorly) it supports human aspcts, and some key exemplar application domains for our work. Finally, Section 8 concludes this paper.

2 MOTIVATION AND RELATED WORK

2.1 Motivating Example: A Smart Home for Ageing

Consider a "smart home" aimed at providing ageing people with technology-based support for physical and mental challenges so they are able to stay in their own home longer and feel safe and secure [7, 20, 17]. To develop such a solution, the software team must deeply understand technologies like sensors, data capture and analysis, communication with hospital systems, and software development methods and tools. However, they must also deeply understand and appreciate the human aspects of their stakeholders: ageing people, their families and friends, and clinicians/community workers. These include the *Technology Proficiency and Acceptance* of ageing people – likely to be much older than the software designers. The development of "Smart homes" technology should factor in the *Emotional* – both positive and negative – reactions to the smart home e.g. daily interaction is potentially positive but being monitored potentially negative. The *Accessibility* of the solutions for people with e.g. physical tremors, poor eyesight, wheel-chair bound, and cognitive decline. Within this, personality differences may be very important e.g. those wanting flexible dialogue compared

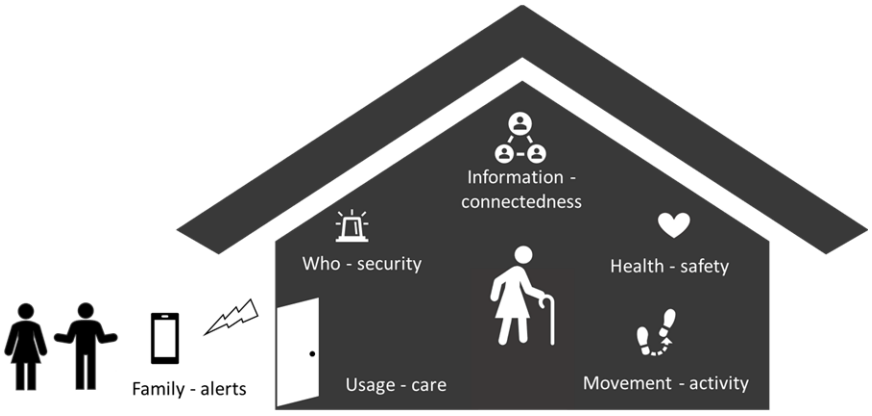

Fig. 1. Simple example of a smart home to support ageing people.

to those needing directive dialogue with the system. The *Usability* of the software for a group of people with varied needs e.g. incorporating the use of voice or gestures or modified smart phone interface. Figure 1 shows an example of such a smart home.

The ageing population is diverse and therefore smart home technology must accommodate for the different *Ages, Genders, Cultures* and *Languages* of users including appropriate use of text, colours, symbols. This is particularly important as one quarter of the elderly in Australia are non-native English speakers and the majority women, but by far the majority of software developers are 20-something years old English-speaking men [32]. Failure to incorporate human aspects into the development of Smart Home software has the potential to result in a home that is unsuitable for who it is designed to help, by introducing confusing, possibly unsettling and invasive, and even potentially dangerous technology.

2.2 Model-driven Software Engineering

Key aspects of Model-Driven Software Engineering (MDSE) are outlined in Fig. 2. MDSE captures high-level models about software requirements i.e. what users need their software to do (a). MDSE then refines these models to detailed designs about how the software solution is organised, composed and its appearance (b). Model transformation then turns these models into software code (c). This is in contrast with most current software development methods which use informal and imprecise models, hand-translation into code via error-prone, and time-consuming low-level hand-coding. Advantages of MDSE-based approaches include capture of formal models of a software system at high levels of abstraction, being able to formally reason about these high-level models and more quickly locate errors, and being able to generate lower-level software artefacts, such as code, without overheads and errors of traditional hand-translating in-

formal models. However, most MDSE approaches use generic requirements and design languages e.g. the Unified Modelling Language (UML) and extensions [11, 33]. These have the disadvantages of being overly complex and are very difficult to use by non-software engineering domain experts [25]. Further MDSE limitations include the often very high level of abstraction of incorporated DSVLs. Necessary customisations or configurations to suit concrete requirements result in the need to implement code-level extensions of the underlying model transformation approach and the code generator.

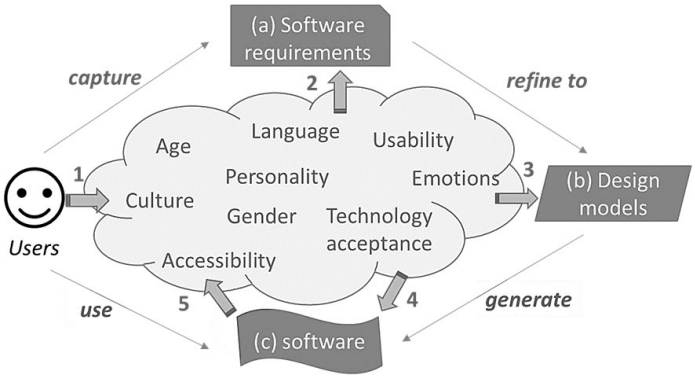

Fig. 2. Incorporating "Human-centric" software issues into Model-Driven Software Engineering (from [19]).

2.3 Domain-specific Visual Languages

Domain-specific Visual Languages (DSVLs) provide a more accessible approach to presenting complex models for domain experts [56]. DSVLs use one or more visual metaphors, typically derived from the domain experts, to represent the model(s). They enable domain experts to understand and even create and use the models directly, rather than rely on software engineers. These DSVLs are then used to generate software code and configuration artefacts to realise a software solution via MDSE approaches. This approach provides higher abstractions and productivity, improves target software quality, provides for repeatability, and supports systematic reuse of best practices [56, 25, 33, 52].

There are many DSVLs for MDSE tools [56, 39, 2]. A representative example is shown in Fig. 3: (a) a custom DSVL designed for clinicians is being used to model a new patient care plan for diabetes and obesity management. Then (b) a model transformer takes the care plan and generates a mobile app to assist the patient to implement their plan [36]. This is a major improvement on developing software using conventional techniques. However, the approach fails to model or incorporate into the mobile app a range of critical human

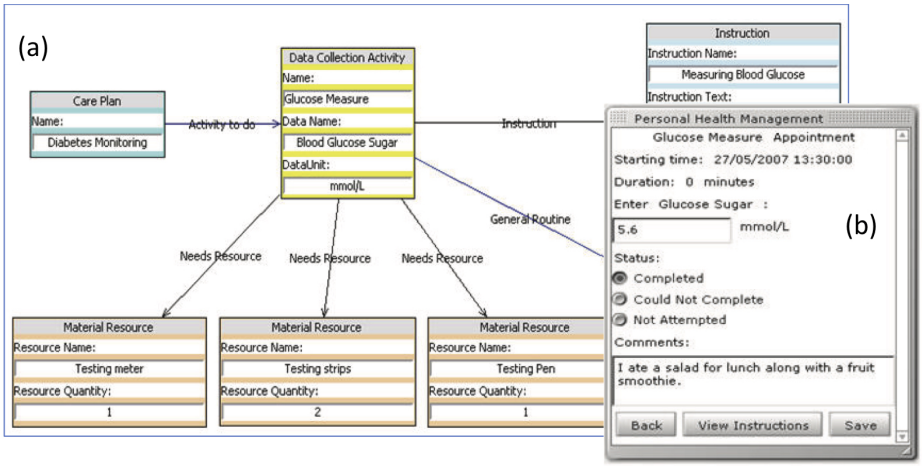

Fig. 3. A clinician-oriented Domain-specific Visual Language for care plan modelling and using Model-driven Engineering to generate an eHealth app ((c) IEEE, from [36]).

aspects, resulting in its failure in practice. Patient-specific, human-centric needs are not captured e.g. technology acceptance and emotional reactions e.g. some patients react negatively to the remote monitoring approach used. Some users are not proficient in English and hence need labels and inputs in their preferred language. Our evaluation found that some of the colours and care plan model language used in the app are confusing for many older users. Users with eyesight limitations find the app too hard to see and too fiddly to interact with. The app can not adapt to different contexts of use or preferences of the users e.g. it can not use their smart home sensors or each patient's particular mobile app dialogue preferences. The app displays a euro-centric terminology about well-being, which may put off some users who prefer e.g. a Buddhist, Confucius or Pacifika view of health concepts from following their care plan [29]. We need to incorporate these human aspects into MDE [4].

2.4 Human Aspects of Software

There has been increasing interest in the human aspects of complex software and how to better incorporate and support these during software development. Agile methods, design thinking and living lab approaches [24, 18, 26, 12] all try and incorporate a human element both in eliciting software requirements and in involving end users of software in the development process [9, 26, 45]. However, none capture human aspects in any systematic way and therefore the software fails to address several critical aspects of the human users. Some new approaches have tried to capture limited human-centric software issues. Emotional aspects of software usage include identifying the emotional reactions of users e.g. when engaging with health and fitness apps or for gaming. Work has been done mod-

elling these Emotional Requirements and applying them to challenging eHealth domains [41, 7].

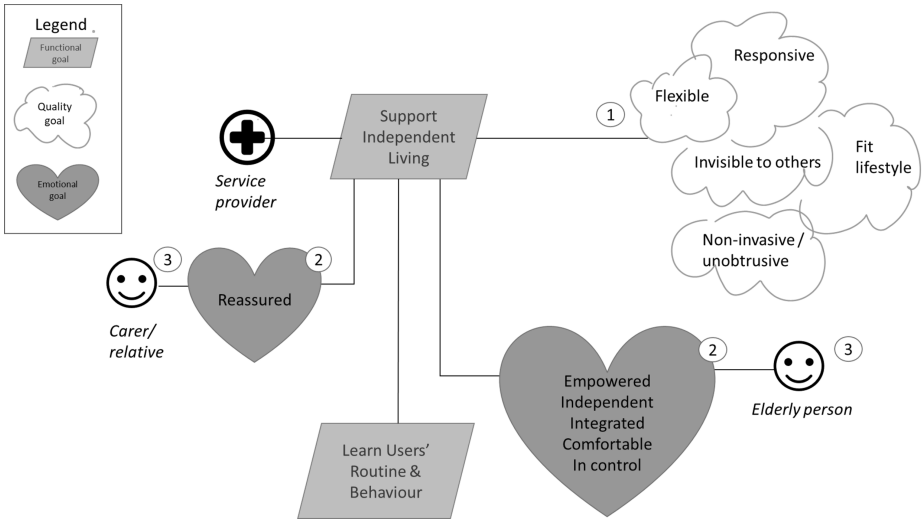

Fig. 4. A Human-centric, Emotion-oriented Domain-specific Visual Language (from [19]).

Fig. 4 shows a representative example using an emotion-oriented requirements DSVL to design better smart homes [20]. Here a conventional goal-based DSVL (1) has been augmented with a set of "emotional goal" elements (2) specific to different users (3). Human characteristics like age, gender, culture and language can dramatically impact aspects of software, especially in the user interface presented by the software and the dialogue had with the user [23, 58, 60]. Limited support for the capture of some of these has been developed. Another example is a multi-lingual requirements tool providing requirements modelling in English and Bahasa Malaysia, including supporting linguistic and some cultural differences between users [31].

Usability testing has long been studied in Human Computer Interaction (HCI) research and practice. However, usability defect reporting is very under-researched in the context of software engineering [62]. Similarly, a lot of work has been done on accessibility in HCI e.g. sight, hearing or cognitively impaired [58, 60], and health IT e.g. mental health challenges when using mobile apps [8]. However, little has been done to evaluate the extent to which physical and mental challenges are properly addressed in engineering software development, and is also poorly supported in practice. Personality, team climate and organisational issues relating to people have been heavily researched in Management, Information Systems, and the personality of programmers and testers in software development [46, 55]. However, little attention has been paid to how to

go about supporting differing personality, team climate or organisational or user culture in software, nor to capture requirements relating to these human aspects. Traditional software requirements and design models have very limited (or no) ability to capture these sorts of human-centric software issues, and approaches are ad-hoc, inconsistent, and incomplete.

Software is fundamentally produced by people, for people. People - and the organisations they work for or that provide them with services - inherently have a set of "values", which differ from person to person and organisation to organisation. Values represent the guiding principles that influence our decision-making processes as individuals, groups and organisations; and they describe what an individual or a group thinks is valuable or important [13]. Such values include but are not limited to openness, transparency, competitiveness, privacy, accessibility, inclusivity, independence, politeness, ambition, respect for authority, and so on. Some approaches have been developed to specify some human values and their relationship to software engineering methods and teams [10].

Current software engineering processes lack consistent, coherent ways to address this range of increasingly important human-centric software issues and thus they are often very incompletely supported or in fact are usually ignored [59, 19]. To date only isolated human aspects have been addressed and often confined to one phase of software development. There are no modelling principles, DSVL-based model design principles, nor widely applicable, practical modelling tools to capture human-centric software issues at requirements or design levels. While a DSVL provides a more human-centric engineering approach, it fails to capture and support the key human aspects in the target software itself. Current MDSE tools, while providing significant software engineering benefits do not support modelling and using these critical human aspects.

3 OUR APPROACH

Fig. 5 illustrates the new human-centric, model-driven software engineering approach we are working to produce. We have identified a set of key approaches that are needed to achieve this vision. We aim to employ several innovative approaches to (i) systematically capture and model a wide range of human-centric software requirements and develop a novel integrated taxonomy and formal model for these; (ii) promote a wide range of human-centric requirements for first-class consideration during software engineering by applying principles for modelling and reasoning about these human-centric requirements using DSVLs; (iii) support a wide range of human-centric requirements in model-driven engineering during software generation and run-time reconfiguration via MDSE techniques; and (iv) systematically use human-centric requirements for requirements-based software testing and reporting human-centric software defects. This approach improves model-driven software engineering by placing crucially important, but to date often forgotten, human-centric aspects of software as first-class considerations in model-driven software engineering. The critical importance of this is really only just becoming recognised, due to the increasing breadth of uses of IT

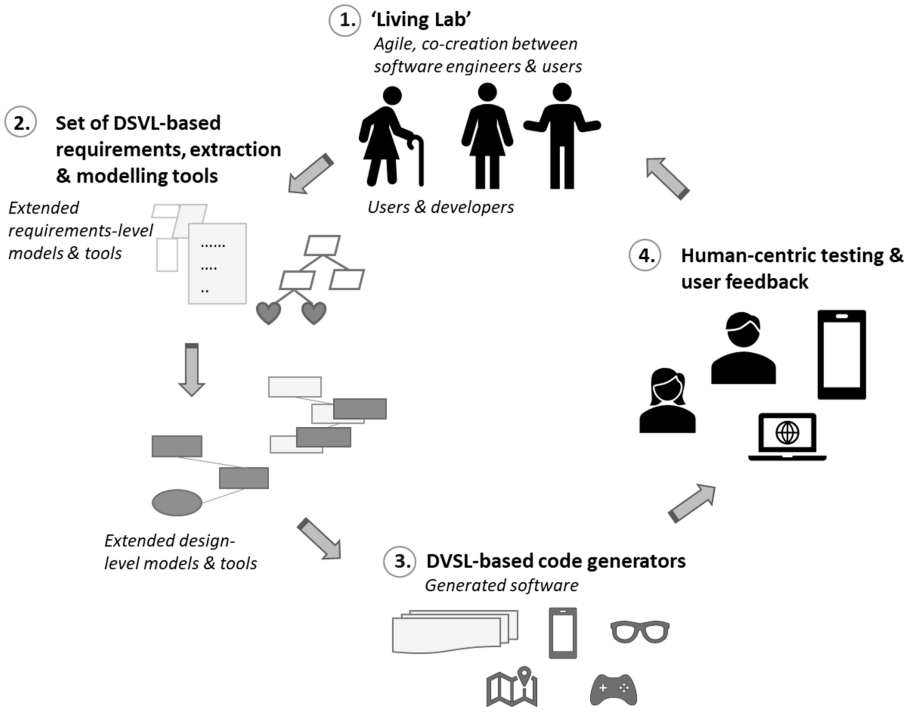

Fig. 5. Our overall HumaniSE approach.

in society and the increasing recognition that understanding and incorporating the very diverse needs of our very diverse software end users is essential. Key features of this approach include:

(1) An Agile, Living Lab approach is being used to co-locate the software team and target end users [20]. This provides a co-creational environment to elicit human-centric requirements, model and capture with human-centric DSVLs, and receive continuous feedback from users. The Living Lab concept's design thinking, agile, co-creation and continuous feedback mechanisms are critical. These are then used to provide an MDSE approach in which human-centric requirements can be effectively and efficiently captured, treated as priorities by the software team, users can quickly report defective software violating these human-centric requirements, and the software team can work effectively with these end users to co-design changes.

(2) A new set of DSVL tools are being developed to capture and model the human-centric requirements, validate them against design principles and best practice modelling patterns, and translate them to extended design-level models. A set of principles for domain-specific visual modelling languages is being developed that enables software engineers to better capture a wide range of human-centric aspects of software: including user's age, gender, cultural preferences,

language needs, emotional needs, personality and cognitive characteristics, and accessibility constraints, both physical and mental. These and other human end user characteristics are essential to prioritise during both software development and software deployment to ensure a useful and usable end product results for a broad range of end users. These principles are used to design a range of novel DSVLs that fully support the capture of many of the important human-centric aspects and model them as the critical requirements issues that they are.

(3) A set of MDSE-based code generators are used to generate software applications – code, configurations, etc. Augmented design models are used to ensure modelled human-centric requirements are preserved for use at design-time to ensure that MDSE-based solutions take them into account appropriately when generating software applications. Unlike existing generators, our extended MDSE generators take into account variations of end-users as specified in the human-centric requirements, producing either multiple versions of the target software applications and/or reconfigurable applications that adapt to each end user's differing human-centric needs.

(4) A combination of human-centric requirements testing and continuous defect feedback are fed to the development team. We are developing a new framework for human-centric, requirements-based testing of software that can verify whether the constructed software systems meet these critical human-centric requirements. By leveraging the Living Lab concept, this enables both faster feedback and defect correction, but also better evolution and modelling of the human-centric requirements over time. Lessons are fed into the improvement of the DSVL tools, best practice patterns and MDSE generators.

Ultimately we want to translate our learning into industry practices and Software Engineering education. To do this we are working with several industry partners, our students, and colleagues teaching Software Engineering courses.

In the following sections, we are explaining these four key features in the format of different projects we are working on across human-centric agile Living Lab, human aspects in requirements engineering, using human aspects in design and implementing software, and evaluating and applying human aspects in software engineering.

4 A HUMAN-CENTRIC AGILE LIVING LAB

Human-centric requirements have to be elicited from target end users (or stakeholders), captured (or modelled) using our DSVL-based tools, used by extended MDSE solutions to generate software, and then the software tested and user feedback accepted and actioned to correct requirements and design model problems. A new approach is needed to effectively support the software team in achieving this. We are investigating the *Living Lab* co-creation concept that has become popular in digital health software development [26, 20].

We are establishing this lab with a domain-specific focus with partner companies and target end users and the software team co-located as in Agile customer-in-team approaches [9, 46]. Target end users and developers closely collaborate to

elicit, capture, test, use and refine the human-centric software requirements. The DSVL modelling tools, MDSE generators and testing tools all need to support collaborative capture, discussion and refinement of the human-centric requirements for this to be most effective. We plan to do this by extending our current work on developing digital health technologies [20], human-centric software engineering processes in software teams, including personality and team climate [49], and collaborative DSVL-based modelling tools [16].

In the following subsections we describe some of our projects that aim to make this living lab for Human-centric Software Engineering a reality.

4.1 Review of Human Aspects in Other Disciplines

We are conducting reviews of the notion of "human aspects" and how they are studied in other disciplines outside software engineering. This includes HCI/UX, information systems, business, design, engineering, psychology, sociology, anthropology, etc. Our objective is to learn from existing bodies of knowledge and to apply relevant theories, notions and findings to build a more complete understanding of human aspects and their implications on software engineering practices. The expected outcome of this project is a more complete, useful and practical taxonomy and ontology of human aspects for use in software engineering.

4.2 Review of how Human Aspects impact Developers

We are conducting reviews of software engineering research literature to better understand - (i) what human aspects have been studied in software engineering to date (ii) how these issues inter-relate and impact software engineer performance; and (iii) where there are key gaps, limitations and need for further studies of human aspects impact on software engineers. From this we aim to determine the range of ways explored to date of how human aspects impact software engineering teams. We then plan to conduct our own studies of under-researched human aspects on software engineers.

4.3 Survey of how Developers Currently Handle End User Human Aspects

To complement the review of works done to understand human aspects impacting software engineers, we are conducting a survey of developers and follow-up interviews to better understand: (i) what are the key human aspects that they encounter when developing software, especially for "challenged" end users (ii) what are the more common, challenging to elicit, challenging to address human aspects for their end-users (iii) how do they currently meet these challenges (iv) are their current best practices we can learn from and disseminate to the wider software engineering community; and (v) what are key practice gaps and challenges that need further R&D to address. From the outcome of this survey and

interview study we plan to focus the work described in the following sections on the particularly important and difficult under-supported end user human aspects for software development.

4.4 Analysis of how Human Aspects of Software are Currently Discussed by Software Engineers

Software development and issue tracking systems such as GitHub, Stack-overflow, Atlassian Jira, Bugzilla, etc provide tools for developers to discuss bugs and development related issues with each other. However, whether human aspects are getting discussed among hundreds of issues developers discuss together, is questionable. On the other hand, software users leave reviews for the apps to share their issues and experiences in using the apps with the other users and developers. Issue tracking and reporting software has many uses for customer service teams, one of which is bug reporting and fix tracking. This software is meant for internal bug tracking, so when team members find bugs and issues while testing products, they can report it to product development.

We are working on mining software repositories and app reviews to better understand whether human aspects are discussed in these platforms. We are also interested to explore what issues developers currently discuss or do not discuss about human aspects in software engineering and how they are currently talked about. At the same time, we are interested in what human aspects do users discuss in app reviews, and how they discuss them. These would enable us to analyse the differences in the discussions across human aspects in software engineering, how the discussions vary between different platforms, e.g. Stack Overflow, GitHub, how discussions vary based on human factor, project, person, etc, and how discussions vary in different fields and applications.

Analysis from this data collection will give us better insights into how discussions vary between developers and users, whether developers address human aspects discussed by the users, and finally, whether developers address what they discuss about the human aspects in software development.

4.5 A Taxonomy of Human-centric Software Requirements

To the best of our knowledge, no taxonomy of human-centric software requirements or even informal definition exists at this time. We are working on developing a new, rich taxonomy of human-centric requirements for software systems. The taxonomy includes different human-centric concepts relating to computer software, and draws on other disciplines including HCI, usability, psychology, sociology, and others to build the conceptual model, and provide detailed relationships and trade-offs between different human-centric requirements.

We are applying this to a number of representative requirements examples to test and refine it, and use the outcomes to inform the development of DSVLs, DSVL tools and MDSE solutions in other activities. This is critical research as it provides software engineers with a lexicon, a set of principles and conceptual model to model and reason about these kinds of requirements. We are conducting

a detailed analysis of several representative real-world software applications from eHealth apps [18], smart homes [20], community service apps [21], educational apps [1], and other heavily human-centric requirements critical domains.

From these we are developing a framework and model for prioritising human-centric software issues. This characterises complex trade-offs and other relationships between different human aspects that make supporting one issue problematic for other issues, similar to the Cognitive Dimensions framework [15]. We plan to use a set of focus groups comprising end users and developers to refine and validate our taxonomy. The taxonomy is being tested on real-world example requirements to gain feedback from both developers and end users to demonstrate its effectiveness. We are drawing on our extensive previous work developing taxonomies for design critics [2], emotion-oriented requirements [7], usability defects [62], and team climate [55].

5 HUMAN ASPECTS IN REQUIREMENTS ENGINEERING

5.1 Extracting Human Aspects from Requirements

Software requirements need to be elicited from end users and these are typically held in a variety of documents and can be obtained in a variety of ways. In the context of the Living Lab we are developing new tools to enable extraction of diverse human-centric requirements from diverse sources, including Powerpoint, Word, Excel, PDF, audio transcripts, images and video. A number of works have addressed different parts of this problem, including extracting requirements using light weight and heavy weight natural language processing [3, 38]. However, none have specifically addressed the extraction of a wide range of human-centric requirements. We are developing, trialing and will then refine a set of extraction tools leveraging existing approaches but focused on human-centric requirements capture and representation using our DSVLs, within our living lab approach, and leveraging our human-centric requirements taxonomy. These tools will also be refined as these other related activities are refined and extended, and applying these tools to representative real-world requirements artefacts will help us to test and extend the outputs of these other tasks. We are focusing on developing leading-edge tools for extracting requirements for goal-directed and multi-lingual models [38, 31], and requirements checking and improvement [3].

5.2 Human Aspects Impacting Requirements Engineers in Agile Teams

We are interested in a range of human aspects in the requirements engineering (RE) domain, particularly those impacting agile requirements engineering teams. These include but are not limited to: (i) how do requirements engineers handle requirements changes during agile software development, from both technical and behavioural/emotional reaction perspectives; (ii) how are agile requirements defined, talked about, is there a taxonomy of “agile” requirements changes;

(iii) how do human aspects impact requirements engineering team members and stakeholders; and (iv) how can we improve RE practices and outcomes by better understanding and taking into account human aspects of team members and stakeholders. To this end we are carrying our studies with RE teams to better understand these issues, design techniques and tools to better evaluate human aspects impacting on RE processes and outcomes, and trial these with partner organisations.

5.3 How are Human Aspects Discussed in Requirements Engineering Documents

Requirements elicitation and specification play an important role in the software development life cycle. Human aspects are often neglected in the early stages of development, i.e., requirements engineering. If human aspects are not taken into account from the early stages of the software development, these issues can impact the final product and make it not tailored toward the diverse range of end-users. Not taking human-centric requirements of users into account can lead to serious impacts to the software under development.

We are working on initially analysing existing requirement engineering documents to explore whether human aspects are discussed/noted, including epics, user stories, use cases, discussion transcripts, feature outlines, and so on. Using the taxonomy of human aspects discussed earlier, we aim to develop guidelines and tools using Natural-Language Processing and Machine Learning techniques to identify relevant human aspects in requirements specifications. This will lead to an improved human aspects-driven requirement engineering process, and an automated tool for identifying human aspects in system artefacts and guiding analysts in deliberately considering these issues during the requirements engineering phase.

5.4 New DSVLs to model Human-Centric Requirements

While DSVLs have been an active research area for at least 20 years, remarkably few principles exist for design and evaluation of effective DSVLs [43]. We are developing a set of new design principles and associated DSVL evaluation approaches to provide more rigorous principles and design steps for specifically human-centric DSVL development. This will require us to identify a range of human-centric software requirements and design issues identified in the taxonomy built. We need to determine how we can best model these, use appropriate visual metaphors to represent the models, how we can support interaction with the visual models, and how we can reason about the suitability of these visual models in terms of usability and effectiveness. We are drawing upon the work on DSVL design tools to achieve this [16, 39, 2], as well as work on 'Physics' of Notations [43] and Cognitive Dimensions [15] to develop these DSVL design principles for modelling human-centric software issues.

We are developing a range of new and augmented DSVLs to model a wide variety of human aspects at the requirements level for software systems. Some of

these DSVLs extend existing requirements modelling languages – in successively more principled ways than currently – e.g. goal-directed requirements languages such as i*, use cases and essential use cases, target user personas, user stories, etc. However, others may provide wholly novel requirements modelling techniques and diagrams that are then linked to other requirements models. We envisage novel requirements capture for things like identifying cultural, age, accessibility and personality aspects of target end users. Where multiple target end users for the same software application have differing human-centric requirements, multiple or composite models may be necessary. Even partial progress here will be very useful for both researchers and practitioners well beyond the scope of our research.

We are building on a wide range of DSVLs, including for design tools, requirements, reporting, business processes, surveys, performance testing, and many others [2, 16, 31] as well as digital health software [20] and work on modelling usability defects and emotional and multi-lingual requirements [7].

5.5 Capturing Human Aspects with Personas

We are investigating the use of personas in requirements engineering with a view to working out ways to better use these during software engineering. To this end we are looking into (i) how personas are currently used in RE and SE; (ii) how to build personas that represent effectively a wide range of human aspects; (iii) how to validate these personas; and (iv) how to use these personas during design and evaluation of software systems. This may include improved ways of defining personas, incorporating specific human aspects into personas, generating personas, and using persona models to support RE, design and evaluation.

6 USING HUMAN ASPECTS IN DESIGN AND IMPLEMENTING SOFTWARE

6.1 Software Design Decision Support

The objective of this project is to develop a decision-support system that systematically guides software developers through capturing selected human aspect needs and requirements. It aims to give developers better support for incorporating these aspects into the design of a software system. Design decisions, contextual information and other tacit information such as design rationales are planned to be formalised using techniques such as decision trees, Markov chains and Bayesian networks. The expected outcome is a demonstrable prototype implementation that can be used and evaluated by software developers. Software developers are humans, and they, therefore, might be subject to cognitive biases and other human challenges. We are studying and evaluating how such a decision-support system may assist developers to use more of System 2 or rational thinking in design [57].

6.2 Collaborative Human-Centric Domain-Specific Visual Languages

We aim to develop a collaborative browser-based domain specific visual languages platform for designing a variety of software tools and systems including data analytics applications, eHealth apps, etc. Multidisciplinary teams of users can design their applications based on their specific characteristics, such as age, gender, culture, personality, etc. Different users can work collaboratively at the same time through a browser-based drag-and-drop based tool in a visualised and programming-free way. Users will be able to store data in, for example, graph databases to enable them to get more specialised views based on their needs and preferences. Domain specific visual languages can be built on top of the existing model-driven approaches such as BiDaML [34], big data analytics modelling languages, and can be further extended by incorporating human-centric issue into the development of the notations. We are adding to our tool code and report generators to automatically generate source code, reports and documents from the visualised models.

6.3 Extending Design Models to include Human Aspects

Model-driven software engineering tools typically use the Unified Modelling Language (UML) or similar design models. Even those using their own design models need to refine higher-level abstract requirements models into lower-level architectural, software design, interface, database and other models. We are working on different ways to effectively extend design-level models to capture necessary design-level human-centric properties, derived from higher level human-centric requirements-level properties [4]. For example, we want to capture design alternatives to achieve an application user interface for a target end user who has sight-impairment, prefers a gesture-based sensor interface to using a Smart phone, has limited mobility, and is quite "neurotic" about device feedback.

We are going to evaluate these different modelling solutions via our living lab with both software engineers and end users, in terms of needed design information and preserving critical human-centred end user needs respectively. Even partial successful outcomes here will be immediately of interest and applicable for software teams. We are extending design models with aspects [44], goal-use case model integration [38], and goal-models extended with emotions [7, 20].

6.4 Human-Centric Design Critics and Modelling Patterns

Just because we add human aspects to our requirements and design models does not mean they may be correct or even appropriate. We are developing a set of proactive tool support systems to advise software engineers of errors or potentially incorrect/unintended issues with their models [2, 48]. This will enable the DSVL toolsets for human-centric requirements and design models to provide proactive feedback to modellers. To enable these design critics are identifying a range of "human-centric requirements and design patterns". These will provide

best-practice approaches to modelling complex requirements and design models mixed with human aspects. These features will be added to successive iterations of our prototype tools from above. This work is building on our approaches to develop DSVL design critics [2] and DSVL-based requirements and design pattern modelling tools [30].

6.5 Using Human Aspects in Model Driven Engineering

Once we have some quality design-level human aspects in models – incremental outcomes from the above activities – we can use these in model-driven engineering code and configuration generators. This research involves adding generators that consume design level models augmented with human-centric properties and synthesizing software applications that use these appropriately. For example, we might generate a gesture-based, passive-voice feedback solution for the target user from the smart home example described in Section 2. However, we might instead generate several interfaces for the same software feature,and at run-time configure the software either with pre-deployment knowledge,end user input, or even modify it while in use based on end user feedback. Thus for example a part of the software for our smart home example could adapt to different end users' current and changing needs (e.g. age, culture, emerging physical and mental challenges, personality etc).

This work is being done incrementally, focusing on single issues first then looking at successively more complex combinations, adding support to the prototype tools and repeatedly trialling the tools. We are adding human aspects to MDSE code generators [4], generating adaptive user interfaces [37], adaptive run-time software [42], and DSVL-based MDSE solutions [56].

7 EVALUATING AND APPLYING HUMAN ASPECTS IN SOFTWARE ENGINEERING

We are addressing critically important issues of (i) testing whether the resultant software generated from our augmented MDSE approach actually meets the requirements specified; (ii) providing a feedback mechanism for end users to report defects in the software specifically relating to human aspects; and (iii) providing a feedback mechanism from software developers to users about changes made relating to their personal human aspects. We are developing human-centric requirements-based testing framework, techniques and tools. These enable human aspects to be used in acceptance tests to improve validation of software against these requirements. We are also developing new human-centric defect reporting mechanisms and developer review and notification mechanisms. These support continuous defect reporting, correction, and feedback via the living lab and remotely. Even partial outcomes would be of immediate benefit to the software engineering research and practice communities. This work is extending research on software tester practices and usability defect reporting [62, 14] and requirements-based testing [31].

7.1 How Can We Provide Better Fixes for Human Aspect-Related Defects

We are working on characterising a mobile app model with the desired human values for its target end-users. We are then using these values to assist us in detecting what we term "values-violating defects" [59] in the target mobile apps. We then provide app developers with a set of recommendations for suggested fixes for these values-violating app defects.

This involves studying a large number of apps, their reviews, and how users feel various "human values" – such as transparency, integrity, privacy, trust, and other human values – may be violated by the apps. We are then identifying ways to (i) detect these "values violations", or values defects, in apps; (ii) identify possible fixes for these defects so that the human values are supported; and (iii) providing tools to developers to help them find and fix these values-related app defects. We hope to generalise this approach to other end user human aspects that need to be supported in apps, including accessibility, age and gender bias, and different end user language and culture.

7.2 Gender Bias in IT Job Ads

We are investigating whether gender is a crucial factor to be an IT professional. As a starting point, we are reviewing if IT job advertisements are more appealing to male candidates. We are using an automated word based gender bias checker to examine if there is any bias in IT job advertisements. We are also conducting a survey of IT hiring managers and IT professionals and/or IT candidates to collect their perception about gender bias in IT job advertisements. Finally, we are applying a cognitive walkthrough approach with gender based persona, proposed by GenderMag [6] tool, to find if male and female candidates react differently to IT job advertisements. The overall finding from this study will help us to identify if IT job advertisements are gender biased, and if yes, what areas need improvement to make those gender inclusive. This will address a critical human aspect where the software engineering profession lacks diversity.

7.3 How Age Affects Users' Interaction with Software

We are looking to generalise the GenderMag [6] approach to supporting other human aspects during evaluation. In this project we are designing new persona templates that include "facets" for different ages that have been shown to influence differently-aged users' interaction with technology. These templates will be customisable using different descriptions of the facets and will generate different persona representing users of different age groups. These enhanced age-related personas can then be used by software engineers to enhance requirements engineering, design and evaluation of software for ageing people [51], in a similar way to GenderMag. Again, we aim to generalise this work to other human aspects of end users.

7.4 eHealth Applications

We are trialing our approaches with real industry practitioners and organisations for whom human aspects are critical. Our approach is particularly suitable for eHealth applications with end users with challenging human aspects, such as physical or mental disability, English as second language, cognitive decline, very young or old, and needing software to adapt to their changing personal or contextual usage needs. Planned target application domains include digital health apps for community members, community educational apps, government service and transport apps and websites, and smart home and smart building management software.

7.5 How Developers Address Accessibility Issues in Mobile Apps

We are studying how developers address one common class of human aspects – accessibility – in mobile apps, by large-scale analysis of app reviews, change histories, and other associated app development and release information. We hope to identify areas where accessibility issues are well-supported and can be more widely adopted. We also hope to identify problematic accessibility issues for developers and use this to carry out targeted studies to improve its support. With mobile apps becoming increasingly widely used for an increasing number of tasks, those not supporting diverse end user accessibility challenges run a great risk of reducing access of many in our communities to critical services [21].

7.6 Developing better apps with personas

We are exploring human-centric smart city development approaches. One of our case studies is the development of better "smart parking" apps. We are using personas to identify a range of parking app users, informed by review mining and other techniques. We are then using these personas to help evaluate existing apps and to then develop and refine requirements and to evaluate designs and prototypes from a range of human-centric perspectives. The idea is to generalise this approach to other smart living systems that by definition have a wide range of diverse end users. We aim to employ the results of adding human aspects to design models and MDE, as discussed above, to improve development of these apps in the future.

We are also exploring the how personas might be used in eHealth. We will use personas to inform the development of a website and eHealth resources for people who have experienced miscarriage. The personas will be developed from inductive qualitative analyses from extensive interviews with women who have had a miscarriage, partners of women who have had a miscarriage, and health practitioners who support women who have had a miscarriage [5, 28, 40] and will include relevant human aspects including users' personalities, age, background, culture, language, physical and mental challenges, comfort with technology and so on. The personas will inform the look and feel of the website and resources, and will be used to test the website during the development and design phases.

The website and resources will be evaluated through surveys and where possible, interviews with people who access the website, and this will be used to validate the personas.

7.7 COVID-19 Apps

In response to the COVID-19 pandemic, there has been an exponential growth in contact tracing apps worldwide [50]. The apps were created very quickly and and are designed to be used across often disparate groups within a population. We are interviewing COVID-19 app developers and conducting focus groups to explore if human-centric aspects suitably taken into account and if so how, or if not how might they be included in the development. This is particularly important for contact tracing apps given their effectiveness is dependent on a critical mass of people engaged with the app.

7.8 Environmental and Sustainability Software Applications

Coastal communities around the world feel the impact of climate change in the form of rising sea levels. Current observations and future projections indicate that sea levels will be significantly higher in the second half of this century [61] causing more frequent and prolonged flooding that pose "unique challenges to risk management decision processes" [22].

We are planning to create a DSVL and decision-support system in collaboration with bay-side communities of Melbourne, Australia, to enable the modelling and simulation of the impact of flooding events and to support automated risk assessments. The DSVL will support the modelling of the needs of the various affected stakeholder groups, e.g. people who live and work in affected areas while also providing means to model localised contextual information about infrastructure, topology as well as local knowledge. We will be following an iterative human-centred design (HCD) approach to devise and evaluate our human-centric decision support system.

7.9 Human Aspects in SE Education and Practice

As argued throughout this paper, we propose to put humans into the focus of SE. Software should adapt to user needs - not the other way round. To build software to do so, a culture change in the SE industry and the way SE is taught is required. Besides improving concepts and methods, we aim to change some terminology in order to explicitly support a new way of thinking, e.g. through redefining or replacing terms such as 'user', 'stakeholder' and 'requirement'.

The term 'user' does not fully convey the human nature and individual differences between people. The term 'stakeholder' is often too narrowly applied. Indirect stakeholders - those who are affected by software systems used by others - are too often overlooked, e.g. self-driving cars may put the safety of other road users at risk [53], facial recognition systems may misidentify innocent people

as criminals [35]. We believe the term 'people' is better suited to express such socio-technical aspects and to break up traditional technology-centred thinking.

Similarly, the term 'need' may be used to overcome the strict separation between functional and non-functional requirements. This differentiation is not only detrimental as the latter are often treated as less important afterthoughts; it is also incomplete. For example, human aspects beyond usability are typically not covered [27].

Moreover, we aim to change the way SE is taught to better prepare future generations of software engineers for the development of successful human-centric software systems. Social sciences need to play a more prominent role in SE curricula. Students and practitioners need the soft skills required for the work in diverse teams and to more effectively elicit and model diverse requirements and perspectives. Various of the above mentioned research activities will inform our revision of SE education.

8 CONCLUSION

Human aspects that are necessary to incorporate into the development of complex software systems include different end user age, language, gender, ethnicity, physical and mental challenges, personality, socio-economic status, educational attainment, emotional reactions, technology proficiency and so on. We described a motivating example - a smart home to support ageing in place - showing how many of these human aspects of software systems need to be fully understood and incorporated by software engineers. To realise this, we described our current work to advance HumaniSE - Human-Centric Software Engineering. This includes the use of a co-creational living lab to better identify diverse end user requirements. The use of domain-specific visual models to improve capture and reasoning about these characteristics. The use of design thinking, extended design models and augmented model-driven engineering. Improving defect reporting to help developers better understand and fix these human aspect-related issues in their software. We are applying these approaches to a range of domains requiring full support of the diverse human aspects of end users. This includes a range of eHealth applications, smart city applications, education-related software, support for vulnerable community members to access and use government and employment services, and sustainability solutions. We very much welcome approaches to discuss collaboration on some of these directions and projects.

Acknowledgements

Support for this work from ARC Discovery Projects DP170101932 and DP200100020 and from ARC Laureate Program FL190100035 is gratefully acknowledged.

References

1. Abdelrazek, M., Ibrahim, A., Cain, A., Grundy, J.: Vision: mobile ehealth learning and intervention platform. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. pp. 252–256 (2018)
2. Ali, N.M., Hosking, J., Grundy, J.: A taxonomy and mapping of computer-based critiquing tools. IEEE Transactions on Software Engineering **39**(11), 1494–1520 (Nov 2013). https://doi.org/10.1109/TSE.2013.32
3. Ali, R., Dalpiaz, F., Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis. Requirements Engineering **15**(4), 439–458 (2010)
4. Ameller, D., Franch, X., Cabot, J.: Dealing with non-functional requirements in model-driven development. In: 2010 18th IEEE international requirements engineering conference. pp. 189–198. IEEE (2010)
5. Bellhouse, C., Temple-Smith, M., Watson, S., Bilardi, J.: "the loss was traumatic... some healthcare providers added to that": Women's experiences of miscarriage. Women and Birth **32**(2), 137–146 (2019)
6. Burnett, M., Stumpf, S., Macbeth, J., Makri, S., Beckwith, L., Kwan, I., Peters, A., Jernigan, W.: Gendermag: A method for evaluating software's gender inclusiveness. Interacting with Computers **28**(6), 760–787 (2016)
7. Curumsing, M.K., Fernando, N., Abdelrazek, M., Vasa, R., Mouzakis, K., Grundy, J.: Emotion-oriented requirements engineering: A case study in developing a smart home system for the elderly. Journal of Systems and Software **147**, 215 – 229 (2019). https://doi.org/https://doi.org/10.1016/j.jss.2018.06.077
8. Donker T, Petrie K, P.J.C.J.B.M.C.H.: Smartphones for smarter delivery of mental health programs: a systematic review. J Med Internet Res **15**(11) (2013). https://doi.org/10.2196/jmir.2791
9. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology **50**(9), 833 – 859 (2008). https://doi.org/https://doi.org/10.1016/j.infsof.2008.01.006, http://www.sciencedirect.com/science/article/pii/S0950584908000256
10. Ferrario, M.A., Simm, W., Forshaw, S., Gradinar, A., Smith, M.T., Smith, I.: Values-first se: research principles in practice. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). pp. 553–562. IEEE (2016)
11. Fontoura, M., Pree, W., Rumpe, B.: The Uml Profile for Framework Architectures. Addison-Wesley Longman Publishing Co., Inc., USA (2000)
12. Friedland, B., Yamauchi, Y.: Reflexive design thinking: putting more human in human-centered practices. interactions **18**(2), 66–71 (2011)
13. Friedman, B., Kahn, P.H., Borning, A.: Value sensitive design and information systems. The handbook of information and computer ethics pp. 69–101 (2008)
14. Garousi, V., Zhi, J.: A survey of software testing practices in canada. Journal of Systems and Software **86**(5), 1354–1376 (2013)
15. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages & Computing **7**(2), 131–174 (1996)
16. Grundy, J.C., Hosking, J., Li, K.N., Ali, N.M., Huh, J., Li, R.L.: Generating domain-specific visual language tools from abstract visual specifications. IEEE Transactions on Software Engineering **39**(4), 487–515 (April 2013). https://doi.org/10.1109/TSE.2012.33

17. Grundy, J.: Human-centric software engineering for next generation cloud-and edge-based smart living applications. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). pp. 1–10. IEEE (2020)
18. Grundy, J., Abdelrazek, M., Curumsing, M.K.: Vision: Improved development of mobile ehealth applications. In: 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft). pp. 219–223. IEEE (2018)
19. Grundy, J., Khalajzadeh, H., Mcintosh, J.: Towards human-centric model-driven software engineering. In: ENASE. pp. 229–238 (2020)
20. Grundy, J., Mouzakis, K., Vasa, R., Cain, A., Curumsing, M., Abdelrazek, M., Fernando, N.: Supporting diverse challenges of ageing with digital enhanced living solutions. In: Global Telehealth Conference 2017. pp. 75–90. IOS Press (2018)
21. Grundy, J., Grundy, J.: A survey of australian human services agency software usage. Journal of technology in human services **31**(1), 84–94 (2013)
22. Hall, J., Weaver, C., Obeysekera, J., Crowell, M., Horton, R., Kopp, R., Marburger, J., Marcy, D., Parris, A., Sweet, W., Veatch, W., White, K.: Rising sea levels: Helping decision-makers confront the inevitable. Coastal Management **47**(2), 127–150 (2019)
23. Hartzel, K.: How self-efficacy and gender issues affect software adoption and use. Communications of the ACM **46**(9), 167–171 (2003)
24. Hoda, R., Salleh, N., Grundy, J.: The rise and evolution of agile software development. IEEE software **35**(5), 58–63 (2018)
25. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of mde in industry. In: Proceedings of the 33rd international conference on software engineering. pp. 471–480 (2011)
26. Hyysalo, S., Hakkarainen, L.: What difference does a living lab make? comparing two health technology innovation projects. CoDesign **10**(3-4), 191–208 (2014)
27. ISO/IEC: Iso/iec 25010 system and software quality models. Tech. rep. (2010)
28. Jensen, K.L., Temple-Smith, M.J., Bilardi, J.E.: Health professionals' roles and practices in supporting women experiencing miscarriage: A qualitative study. Australian and New Zealand Journal of Obstetrics and Gynaecology **59**(4), 508–513 (2019)
29. Joseph, A.J.: The necessity of an attention to eurocentrism and colonial technologies: An addition to critical mental health literature. Disability & Society **30**(7), 1021–1041 (2015)
30. Kamalrudin, M., Hosking, J., Grundy, J.: Improving requirements quality using essential use case interaction patterns. In: 2011 33rd International Conference on Software Engineering (ICSE). pp. 531–540. IEEE (2011)
31. Kamalrudin, M., Hosking, J., Grundy, J.: Maramaaic: tool support for consistency management and validation of requirements. Automated software engineering **24**(1), 1–45 (2017)
32. Kenny, E.J., Donnelly, R.: Navigating the gender structure in information technology: How does this affect the experiences and behaviours of women? Human Relations **73**(3), 326–350 (2020)
33. Kent, S.: Model driven engineering. In: International Conference on Integrated Formal Methods. pp. 286–298. Springer (2002)
34. Khalajzadeh, H., Simmons, A., Abdelrazek, M., Grundy, J., Hosking, J., He, Q.: An end-to-end model-based approach to support big data analytics development. Journal of Computer Languages p. 100964 (2020)

35. Khalil, A., Ahmed, S.G., Khattak, A.M., Al-Qirim, N.: Investigating bias in facial analysis systems: A systematic review. IEEE Access **8**, 130751–130761 (2020)
36. Khambati, A., Grundy, J., Warren, J., Hosking, J.: Model-driven development of mobile personal health care applications. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. pp. 467–470. IEEE (2008)
37. Lavie, T., Meyer, J.: Benefits and costs of adaptive user interfaces. International Journal of Human-Computer Studies **68**(8), 508–524 (2010)
38. Lee, J., Xue, N.L.: Analyzing user requirements by use cases: A goal-driven approach. IEEE software **16**(4), 92–101 (1999)
39. Li, L., Grundy, J., Hosking, J.: A visual language and environment for enterprise system modelling and automation. Journal of Visual Languages & Computing **25**(4), 253–277 (2014)
40. Miller, E.J., Temple-Smith, M.J., Bilardi, J.E.: 'there was just no-one there to acknowledge that it happened to me as well': A qualitative study of male partner's experience of miscarriage. PloS one **14**(5), e0217395 (2019)
41. Miller, T., Pedell, S., Lopez-Lorca, A.A., Mendoza, A., Sterling, L., Keirnan, A.: Emotion-led modelling for people-oriented requirements engineering: the case study of emergency systems. Journal of Systems and Software **105**, 54–71 (2015)
42. Mohamed Almorsy, John Grundy, A.S.I.: Adaptable, model-driven security engineering for saas cloud-based applications. Automated software engineering **21**(2), 187–224 (2014)
43. Moody, D.: The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on software engineering **35**(6), 756–779 (2009)
44. Mouheb, D., Talhi, C., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: Weaving security aspects into uml 2.0 design models. In: Proceedings of the 13th workshop on Aspect-oriented modeling. pp. 7–12 (2009)
45. Mummah, S.A., Robinson, T.N., King, A.C., Gardner, C.D., Sutton, S.: Ideas (integrate, design, assess, and share): a framework and toolkit of strategies for the development of more effective digital interventions to change health behavior. Journal of medical Internet research **18**(12), e317 (2016)
46. Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J.: The impact of agile practices on communication in software development. Empirical Software Engineering **13**(3), 303–337 (2008)
47. Prikladnicki, R., Dittrich, Y., Sharp, H., De Souza, C., Cataldo, M., Hoda, R.: Cooperative and human aspects of software engineering: Chase 2013. SIGSOFT Softw. Eng. Notes **38**(5), 34–37 (Aug 2013). https://doi.org/10.1145/2507288.2507321, https://doi.org/10.1145/2507288.2507321
48. Robbins, J.E., Redmiles, D.F.: Software architecture critics in the argo design environment. Knowledge-Based Systems **11**(1), 47–60 (1998)
49. Salleh, N., Hoda, R., Su, M.T., Kanij, T., Grundy, J.: Recruitment, engagement and feedback in empirical software engineering studies in industrial contexts. Information and software technology **98**, 161–172 (2018)
50. Samhi, J., Allix, K., Bissyandé, T.F., Klein, J.: A first look at android applications in google play related to covid-19. arXiv preprint arXiv:2006.11002 (2020)
51. Sarcar, S., Munteanu, C., Jokinen, J.P., Oulasvirta, A., Silpasuwanchai, C., Charness, N., Dunlop, M., Ren, X.: Designing mobile interactions for the ageing populations. In: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems. pp. 506–509 (2017)
52. Schmidt, D.C.: Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY- **39**(2), 25 (2006)

53. S.Combs, T., S.Sandt, L., P.Clamann, M., C.McDonald, N.: Automated vehicles and pedestrian safety: Exploring the promise and limits of pedestrian detection. American Journal of Preventive Medicine **56**(1), 1–7 (2019)
54. Smith, J.: The Book. The publishing company, London, 2nd edn. (1998)
55. Soomro, A.B., Salleh, N., Mendes, E., Grundy, J., Burch, G., Nordin, A.: The effect of software engineers' personality traits on team climate and performance: A systematic literature review. Information and Software Technology **73**, 52–65 (2016)
56. Sprinkle, J., Karsai, G.: A domain-specific visual language for domain model evolution. Journal of Visual Languages & Computing **15**(3-4), 291–307 (2004)
57. Stanovich, K., West, R.: Individual differences in reasoning: Implications for the rationality debate. The Behavioral and brain sciences **23**, 645–65; discussion 665 (11 2000). https://doi.org/10.1017/S0140525X00003435
58. Stock, S.E., Davies, D.K., Wehmeyer, M.L., Palmer, S.B.: Evaluation of cognitively accessible software to increase independent access to cellphone technology for people with intellectual disability. Journal of Intellectual Disability Research **52**(12), 1155–1164 (2008)
59. Whittle, J.: Is your software valueless? IEEE Software **36**(3), 112–115 (2019)
60. Wirtz, S., Jakobs, E.M., Ziefle, M.: Age-specific usability issues of software interfaces. In: Proceedings of the IEA. vol. 17 (2009)
61. Wright, L., Syvitski, J., Nichols, C.: Sea level rise: Recent trends and future projections. Coastal Research Library **27**, 47–57 (2019)
62. Yusop, N.S.M., Grundy, J., Vasa, R.: Reporting usability defects: A systematic literature review. IEEE Transactions on Software Engineering **43**(9), 848–867 (2016)