Towards Human-centric Model-driven Software Engineering

John Grundy[a], Hourieh Khalajzadeh[b] and Jennifer Mcintosh[c]
Faculty of Information Technology, Monash University, Clayton, Victoria, Australia

Keywords: Model-driven Engineering, Human-centric Software Engineering, Human Factors.

Abstract: Many current software systems suffer from a lack of consideration of the human differences between end users. This includes age, gender, language, culture, emotions, personality, education, physical and mental challenges, and so on. We describe our work looking to consider these characteristics by incorporation of human centric-issues throughout the model-driven engineering process lifecycle. We propose the use of the co-creational "living lab" model to better collect human-centric issues in the software requirements. We focus on modelling these human-centric factors using domain-specific visual languages, themselves human-centric modelling artefacts. We describe work to incorporate these human-centric issues into model-driven engineering design models, and to support both code generation and run-time adaptation to different user human factors. We discuss continuous evaluation of such human-centric issues in the produced software and feedback of user reported defects to requirements and model refinement.

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 overruns 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) (Curumsing et al., 2019; Prikladnicki et al., 2013; Yusop et al., 2016). This leads to huge economic cost, inefficiencies, not fit-for-purpose solutions, and dangerous and potentially lifethreatening 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 human-centric issues during the software engineering process (Hartzel, 2003; Miller et al., 2015; Stock et al., 2008; Wirtz et al., 2009; Smith, 1998).

[a] https://orcid.org/0000-0003-4928-7076
[b] https://orcid.org/0000-0001-9958-0102
[c] https://orcid.org/0000-0002-6655-0940

2 MOTIVATING EXAMPLE

2.1 Smart Home

Consider a representative example - 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 (Curumsing et al., 2019; Grundy et al., 2018). To develop 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 to those needing directive dialogue with the system.

Grundy, J., Khalajzadeh, H. and Mcintosh, J.
Towards Human-centric Model-driven Software Engineering.
DOI: 10.5220/0009806002290238
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 229-238
ISBN: 978-989-758-421-3

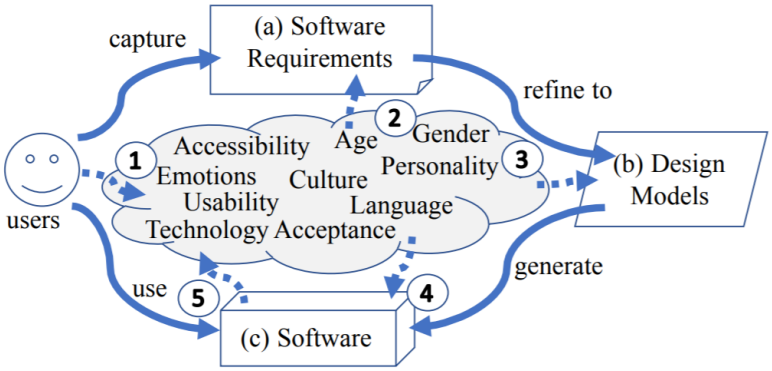

Figure 1: Incorporating "Human-centric" software issues into Model-Driven Software Engineering.

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.

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 year old English-speaking men. Within this, personality differences may be very important e.g. those wanting flexible dialogue compared to those needing directive dialogue with the system. Failure to incorporate human-centric issues 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 Key Challenges

Current software engineering approaches ignore many of these human-centric issues or address them in piece-meal, ad-hoc ways (Prikladnicki et al., 2013; Curumsing et al., 2019; Hartzel, 2003; Wirtz et al., 2009; Ameller et al., 2010). For example, key aspects of Model-Driven Software Engineering (MDSE) are outlined in Figure 1. In MDSE, user requirements for the software are captured and represented by a variety of abstract requirements models (a). These are then refined to detailed design models (b) to describe the software solution. These design models are then transformed by a set of generators into software code (c) to implement the target system (Schmidt, 2006). However, currently almost no human-centric issues are captured, reasoned about or used when producing or testing this software (Miller et al., 2015; Yusop et al., 2016).

2.3 Generating More Human-centric Software

We need to fully integrate these human-centric issues into model-driven software development. In order to do this, we aim to carry out the following steps:

1. Use a co-creational, agile Living Lab-based approach, enabling software teams to capture and reason about under-represented, under-used, under-supported yet critical human-centric requirements of target software;
2. Develop a new set of human-centric requirements modelling languages, enabling software engineers to effectively model diverse human-centric issues, along with 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,...;
3. Augment conventional model-driven engineering design models with human-centric requirements for use during MDSE, along with techniques to verify the completeness, correctness and consistency of these models, and proactively check them against best practice models and principles;
4. Develop new techniques to incorporate these human-centric issues in design models into MDSE-based software code generators, enabling target software to dynamically adapt to differing user needs at run-time; and
5. Use these human-centric requirements during software testing to support human-centric requirements-based testing of software systems, along with techniques to receive feedback from users on human factors-related defects in their software solutions.

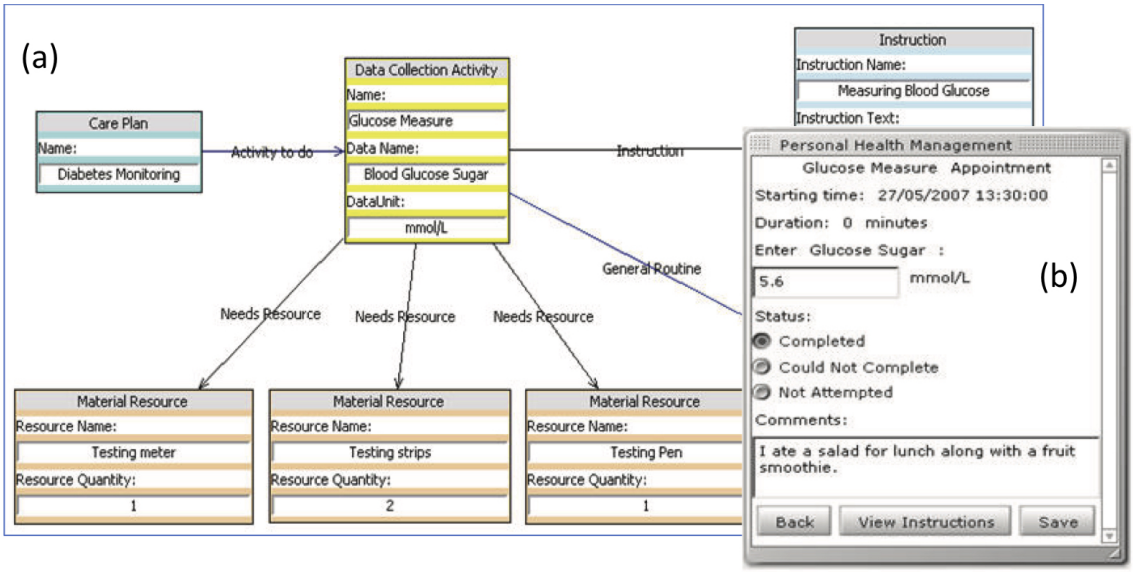

Figure 2: A clinician-oriented Domain-specific Visual Language for care plan modelling and using Model-driven Engineering to generate an eHealth app.

3 BACKGROUND

Model-Driven Software Engineering (MDSE) captures high-level models about software requirements i.e. what users need their software to do. MDSE then refines these models to detailed designs about how the software solution is organised, composed and its appearance. Model transformation then turns these models into software code. 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 informal models.

However, most MSDE approaches use generic requirements and design languages e.g. the Unified Modelling Language (UML) and extensions (Fontoura et al., 2000; Kent, 2002). These have the disadvantages of being overly complex and very difficult to use by non-software engineering domain experts (Hutchinson et al., 2011). Domain-specific Visual Languages (DSVLs) provide a more accessible approach to presenting complex models for domain experts (Sprinkle and Karsai, 2004). 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 (Sprinkle and Karsai, 2004; Hutchinson et al., 2011; Kent, 2002; Schmidt, 2006).

There are many DSVLs for MDSE tools (Sprinkle and Karsai, 2004; Li et al., 2014; Ali et al., 2013). A representative example is shown in Figure 2: (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 (Khambati et al., 2008). 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-centric issues, 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 English speakers. Colours and care plan model language used in the app are confusing for many older users. Users with eye-sight 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 Euro-centric terminology about well-being, therefore, putting off some users from following their care plan. We need to incorporate these human-centric issues into MDE (Ameller et al., 2010).

There has been recent interest in the human-centric issues of complex software and how to better support these during software development. Agile methods, design thinking and living lab approaches all try and incorporate a human element both in eliciting software requirements and in involving end users of software in the development process (Dybå and Dingsøyr, 2008; Hyysalo and Hakkarainen, 2014; Mummah et al., 2016). However, none capture human-centric issues 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 modelling these Emotional Requirements and applying them to challenging eHealth domains (Miller et al., 2015; Curumsing et al., 2019).

Figure 3 shows a representative example using an emotion-oriented requirements DSVL to design better smart homes (Grundy et al., 2018). 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 (Hartzel, 2003; Stock et al., 2008; Wirtz et al., 2009). 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 (Kamalrudin et al., 2017).

Usability and 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 (Yusop et al., 2016). Similarly, a lot of work has been done on accessibility in HCI e.g. sight, hearing or cognitively impaired (Stock et al., 2008; Wirtz et al., 2009), and health IT e.g. mental health challenges when using mobile apps (Donker et al., 2013). 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 (Pikkarainen et al., 2008; Soomro et al., 2016). 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-centric issues. 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.

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-centric issues in the target software itself. Current MDSE tools, while providing significant software engineering benefits do not support modelling and using these critical human-centric issues. Current software engineering processes lack consistent, coherent ways to address these increasingly important human-centric software issues and thus they are often very incompletely supported or in fact are usually ignored.

4 APPROACH

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 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 will improve 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 in society and the increasing recognition that under-

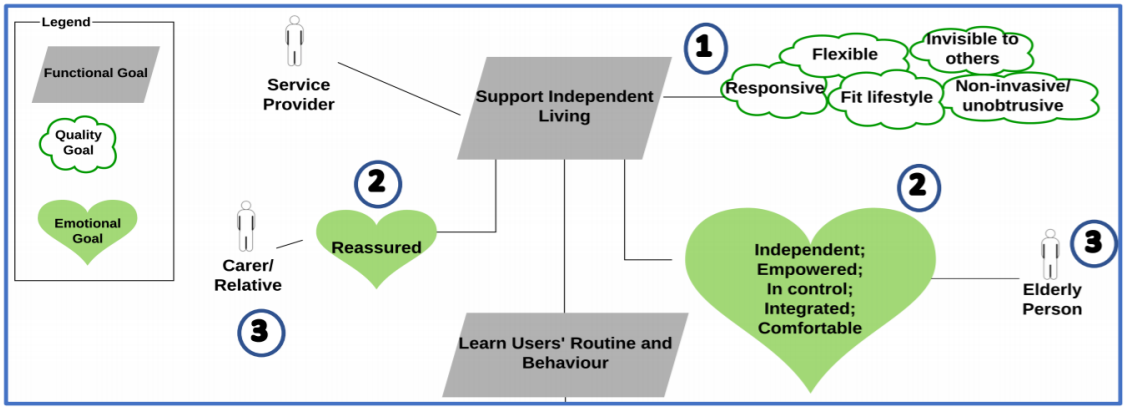

Figure 3: A Human-centric, Emotion-oriented Domain-specific Visual Language.

standing and incorporating the very diverse needs of our very diverse software end users is essential. Key features of this approach include:

- Use of the Living Lab concept's design thinking, agile, co-creation and continuous feedback mechanisms. These will be used to provide a 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.
- A set of principles for domain-specific visual modelling languages will be developed that enable software engineers to 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 will be 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.
- Augmented models that ensure these modelled human-centric properties are preserved for use at design-time to ensure that MDSE-based solutions take them into account appropriately when generating software applications.
- We will develop 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.

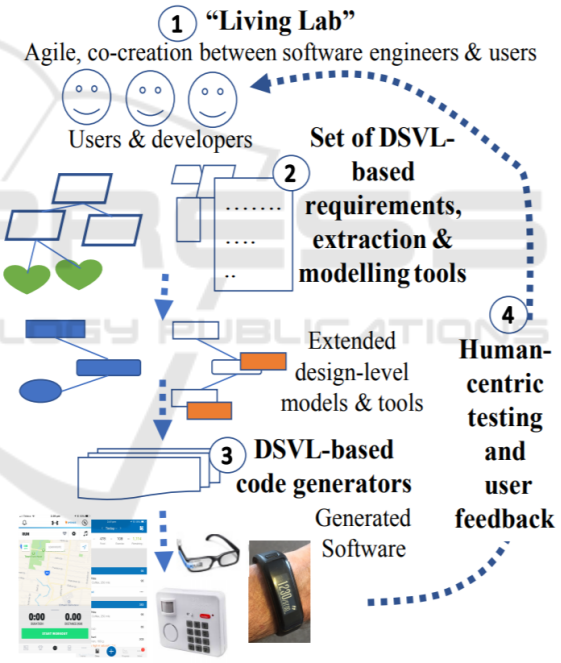

Figure 4: Overall approach.

5 RESEARCH ACTIVITIES

Figure 4 illustrates the new human-centric, model-driven software engineering approach we aim to produce. We have identified a set of key approaches that are needed to achieve this vision. An Agile Living Lab approach will be used to colocate the software team and target end users (Grundy et al., 2018). This will provide a co-creational environment to elicit human-centric requirements, model and capture with

human-centric DSVLs, and receive continuous feedback from users.

A set of DSVL tools will be used 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 MDSE generators will generate software applications – code, configurations, etc. Unlike existing generators, these will 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.

A combination of human-centric requirements testing and continuous defect feedback will be fed to the development team. By leveraging the Living Lab concept, this will enable both faster feedback and defect correction, but also better evolution and modelling of the human-centric requirements over time. Lessons will be fed into the improvement of the DSVL tools, best practice patterns and MDSE generators. We have identified a set of necessary research activities to achieve this outcome, summarised below.

5.1 A Human-centric Agile Living Lab

Human-centric requirements have to be elicited from target end users (or stakeholders), captured (or modelled) using our DVSL-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, and propose investigating the Living Lab co-creation concept that has become popular in digital health software development (Hyysalo and Hakkarainen, 2014; Grundy et al., 2018). 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 (Dybå and Dingsøyr, 2008; Pikkarainen et al., 2008). 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 (Grundy et al., 2018), human-centric software engineering processes in software teams, including personality and team climate (Salleh et al., 2018), and collaborative DSVL-based modelling tools (Grundy et al., 2013).

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 will include different human-centric concepts relating to computer software, and will draw on other disciplines including HCI, usability, psychology, 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 will provide software engineers 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, smart homes, community service apps, educational apps, and other heavily human-centric requirements critical domains.

From these we are developing a framework and model for prioritising human-centric software issues. This will characterise complex trade-offs and other relationships between different human-centric issues that make supporting one issue problematic for other issues, similar to the Cognitive Dimensions framework (Green and Petre, 1996). We will use focus groups with end users and developers to refine and validate our taxonomy. The taxonomy will be 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 (Ali et al., 2013), emotion-oriented requirements (Curumsing et al., 2019), usability defects (Yusop et al., 2016), and team climate (Soomro et al., 2016).

5.2 Human-centric Requirements Engineering

While DSVLs have been an active research area for at least 20 years, remarkably few principles exist for design and evaluation of effective DSVLs (Moody, 2009). 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 for the range of a

human-centric software requirements and design issues identified in the taxonomy built, how we can 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 (Grundy et al., 2013; Li et al., 2014; Ali et al., 2013), as well as work on 'Physics' of Notations (Moody, 2009) and Cognitive Dimensions (Green and Petre, 1996) 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-centric issues 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. We are building on a wide range of DSVLs, including for design tools, requirements, reporting, business processes, surveys, performance testing, and many others (Ali et al., 2013; Grundy et al., 2013; Kamalrudin et al., 2017) as well as digital health software (Grundy et al., 2018) and work on modelling usability defects and emotional and multi-lingual requirements (Curumsing et al., 2019).

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. We will work - in the context of the Living Lab - to develop new tools to extract 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 (Ali et al., 2010; Lee and Xue, 1999). However, none have specifically addressed the extraction of a wide range of human-centric requirements. We will develop, trial and 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 will develop leading-edge tools for extracting requirements for goal-directed and multi-lingual models (Lee and Xue, 1999; Kamalrudin et al., 2017), and requirements checking and improvement (Ali et al., 2010).

5.3 Using Human-centric Issues in Model Driven Engineering

MDSE tools typically use 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 will explore 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 (Ameller et al., 2010). 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 will evaluate the 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 will extend design models with aspects (Mouheb et al., 2009), goal-use case model integration (Lee and Xue, 1999), and goal-models extended with emotions (Curumsing et al., 2019; Grundy et al., 2018). Just because we add human-centric issues to our requirements and design models does not mean they may be correct or even appropriate. We will develop a set of proactive tool support systems to advise software engineers of errors or potentially incorrect/unintended issues with their models (Ali et al., 2013; Robbins and Redmiles, 1998). This will enable the DSVL toolsets for human-centric requirements and design models to provide proactive feedback to modellers. To enable these design critics we will develop 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-centric

issues. These features will be added to successive iterations of our prototype tools from above. This work will build on our approaches to develop DSVL design critics (Ali et al., 2013) and DSVL-based requirements and design pattern modelling tools (Kamalrudin et al., 2011).

Once we have some quality design-level human-centric issues in models (incremental outcomes from the above activities), we can use these in model-driven engineering code and configuration generators. Results from this work will be fed back to extending the taxonomy and human-centric design principles. This will involve 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 example above. 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 will be 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 will work on adding human-centric issues to MDSE code generators (Ameller et al., 2010), generating adaptive user interfaces (Lavie and Meyer, 2010), adaptive run-time software (Mohamed Almorsy, 2014), and DSVL-based MDSE solutions (Sprinkle and Karsai, 2004).

5.4 Using Human-centric Requirements in Testing, Defect Reporting and Continuous Feedback

Here we will address critically important issues of (i) testing whether the resultant software generated from our augmented MDSE actually meets the requirements specified; (ii) providing a feedback mechanism for end users to report defects in the software specifically relating to human-centric issues; and (iii) providing a feedback mechanism from software developers to users about changes made relating to their personal human-centric issues. We are developing a human-centric requirements-based testing framework, techniques and tools. These enable human-centric issues to be used in acceptance tests to improve validation of software against these requirements. We will also develop new human-centric defect reporting mechanisms and developer review and notification mechanisms. These will 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 (Yusop et al., 2016; Garousi and Zhi, 2013) and requirements-based testing (Kamalrudin et al., 2017).

5.5 Trials on Industry-scale Examples

We will trial our approach with real industry practitioners and organisations for whom human-centric issues are critical. Our approach is particularly suitable on applications with end users with challenging human-centric issues, 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.

6 CONCLUSION AND PERSPECTIVES

We have proposed a new approach to incorporating and supporting a wide range of human-centric issues into model-driven software development. We are setting up a co-creational, agile living lab in which to collaborative elicit user requirements, capture in augmented models, generate multiple solutions, and enable requirements-based testing and feedback. We are developing a new taxonomy of human-centric issues and then incorporating these into requirements and design modelling languages and tools. We are using these augmented models in model-driven engineering to generate multiple versions of software to support diverse end user needs, or support run-time adaptation of the generated code to these needs. We are supporting iterative user feedback to proactively incorporate human-centric issues into models and re-generating and re-deploying solutions.

ACKNOWLEDGEMENTS

Support from ARC Discovery Project DP170101932 and ARC Laureate Program FL190100035 is gratefully acknowledged.

REFERENCES

Ali, N. M., Hosking, J., and Grundy, J. (2013). A taxonomy and mapping of computer-based critiquing tools. *IEEE Transactions on Software Engineering*, 39(11):1494–1520.

Ali, R., Dalpiaz, F., and Giorgini, P. (2010). A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458.

Ameller, D., Franch, X., and Cabot, J. (2010). Dealing with non-functional requirements in model-driven development. In *2010 18th IEEE international requirements engineering conference*, pages 189–198. IEEE.

Curumsing, M. K., Fernando, N., Abdelrazek, M., Vasa, R., Mouzakis, K., and Grundy, J. (2019). Emotion-oriented requirements engineering: A case study in developing a smart home system for the elderly. *Journal of Systems and Software*, 147:215 – 229.

Donker, T., Petrie, K., Proudfoot, J., Clarke, J., Birch, M., and Christensen, H. (2013). Smartphones for smarter delivery of mental health programs: a systematic review. *J Med Internet Res*, 15(11).

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859.

Fontoura, M., Pree, W., and Rumpe, B. (2000). *The Uml Profile for Framework Architectures*. Addison-Wesley Longman Publishing Co., Inc., USA.

Garousi, V. and Zhi, J. (2013). A survey of software testing practices in canada. *Journal of Systems and Software*, 86(5):1354–1376.

Green, T. R. G. and Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174.

Grundy, J., Mouzakis, K., Vasa, R., Cain, A., Curumsing, M., Abdelrazek, M., and Fernando, N. (2018). Supporting diverse challenges of ageing with digital enhanced living solutions. In *Global Telehealth Conference 2017*, pages 75–90. IOS Press.

Grundy, J. C., Hosking, J., Li, K. N., Ali, N. M., Huh, J., and Li, R. L. (2013). Generating domain-specific visual language tools from abstract visual specifications. *IEEE Transactions on Software Engineering*, 39(4):487–515.

Hartzel, K. (2003). How self-efficacy and gender issues affect software adoption and use. *Communications of the ACM*, 46(9):167–171.

Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). Empirical assessment of mde in industry. In *Proceedings of the 33rd international conference on software engineering*, pages 471–480.

Hyysalo, S. and Hakkarainen, L. (2014). What difference does a living lab make? comparing two health technology innovation projects. *CoDesign*, 10(3-4):191–208.

Kamalrudin, M., Hosking, J., and Grundy, J. (2011). Improving requirements quality using essential use case interaction patterns. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 531–540. IEEE.

Kamalrudin, M., Hosking, J., and Grundy, J. (2017). Maramaaic: tool support for consistency management and validation of requirements. *Automated software engineering*, 24(1):1–45.

Kent, S. (2002). Model driven engineering. In *International Conference on Integrated Formal Methods*, pages 286–298. Springer.

Khambati, A., Grundy, J., Warren, J., and Hosking, J. (2008). Model-driven development of mobile personal health care applications. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 467–470. IEEE.

Lavie, T. and Meyer, J. (2010). Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies*, 68(8):508–524.

Lee, J. and Xue, N.-L. (1999). Analyzing user requirements by use cases: A goal-driven approach. *IEEE software*, 16(4):92–101.

Li, L., Grundy, J., and Hosking, J. (2014). A visual language and environment for enterprise system modelling and automation. *Journal of Visual Languages & Computing*, 25(4):253–277.

Miller, T., Pedell, S., Lopez-Lorca, A. A., Mendoza, A., Sterling, L., and Keirnan, A. (2015). Emotion-led modelling for people-oriented requirements engineering: the case study of emergency systems. *Journal of Systems and Software*, 105:54–71.

Mohamed Almorsy, John Grundy, A. S. I. (2014). Adaptable, model-driven security engineering for saas cloud-based applications. *Automated software engineering*, 21(2):187–224.

Moody, D. (2009). The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on software engineering*, 35(6):756–779.

Mouheb, D., Talhi, C., Lima, V., Debbabi, M., Wang, L., and Pourzandi, M. (2009). Weaving security aspects into uml 2.0 design models. In *Proceedings of the 13th workshop on Aspect-oriented modeling*, pages 7–12.

Mummah, S. A., Robinson, T. N., King, A. C., Gardner, C. D., and Sutton, S. (2016). 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.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337.

Prikladnicki, R., Dittrich, Y., Sharp, H., De Souza, C., Cataldo, M., and Hoda, R. (2013). Cooperative and

human aspects of software engineering: Chase 2013. *SIGSOFT Softw. Eng. Notes*, 38(5):34–37.

Robbins, J. E. and Redmiles, D. F. (1998). Software architecture critics in the argo design environment. *Knowledge-Based Systems*, 11(1):47–60.

Salleh, N., Hoda, R., Su, M. T., Kanij, T., and Grundy, J. (2018). Recruitment, engagement and feedback in empirical software engineering studies in industrial contexts. *Information and software technology*, 98:161–172.

Schmidt, D. C. (2006). Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25.

Smith, J. (1998). *The Book*. The publishing company, London, 2nd edition.

Soomro, A. B., Salleh, N., Mendes, E., Grundy, J., Burch, G., and Nordin, A. (2016). The effect of software engineers' personality traits on team climate and performance: A systematic literature review. *Information and Software Technology*, 73:52–65.

Sprinkle, J. and Karsai, G. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages & Computing*, 15(3-4):291–307.

Stock, S. E., Davies, D. K., Wehmeyer, M. L., and Palmer, S. B. (2008). 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.

Wirtz, S., Jakobs, E.-M., and Ziefle, M. (2009). Age-specific usability issues of software interfaces. In *Proceedings of the IEA*, volume 17.

Yusop, N. S. M., Grundy, J., and Vasa, R. (2016). Reporting usability defects: A systematic literature review. *IEEE Transactions on Software Engineering*, 43(9):848–867.