A Case for Human Values in Software Engineering

Jon Whittle, Faculty of Information Technology, Monash University, Australia

Maria Angela Ferrario, School of Computing and Communications, Lancaster University, UK

Will Simm, School of Computing and Communications, Lancaster University, UK

Waqar Hussain, Faculty of Information Technology, Monash University, Australia

Abstract

This article argues that human values – such as responsibility, transparency, creativity, and equality – are heavily under-represented in software engineering methods. Based on experiences with real-world projects with not-for-profits, we explore how human values can be integrated into existing participatory agile practices. We propose new ways of considering human values in software practice, including: the use of the Schwartz taxonomy of human values and *values portraits* to contextualise values definitions; the use of values as a way to capture the rationale for requirements to ensure a culture of values throughout the software lifecycle; and a simple adaptation of agile methods to include a role for a 'critical friend' who can champion values during decision making.

Keywords/taxonomy: D.2.14 Human Factors in Software Design, H.1.2.b Human-centered computing, D.2.1 Requirements/Specifications

Three Actionable Insights:

1) Existing social science theories of human values can be brought into requirements engineering to provide techniques for defining and refining human values in a software context.
2) Values provide the 'why' in requirements engineering – by documenting requirements with this rationale, a project can ensure that values are considered throughout the software lifecycle.
3) There is no need for completely new software processes to handle human values; rather, existing processes (e.g., agile methods) can readily be adapted to include consideration of human values.

This article makes the case for considering human values (such as diversity, integrity and sense of belonging) as "first class entities" in software engineering. Software practice includes a wide body of knowledge on how to properly handle technical values (such as security and performance) and business value but is lacking guidelines for how to deal with the broader set of human values. In particular, there

are no systematic software engineering methods that detail how to define, refine, and monitor human values throughout the software development lifecycle.

Recent high profile cases have shown that ignoring human values can have devastating negative economic impact. In the Volkswagen emissions scandal [1], software was deliberately designed in contradiction to the company's corporate value of "responsible thinking", a decision that led to the resignation of the CEO, a 30% drop in VW's stock price and 25% drop in sales within a year. This is not, however, an isolated example. Galhotra et al. [2] cite many examples of 'biased' software systems, which act in a way different to the values intended by their designers: Amazon's software for determining which US zip codes to give free same day shipping, which turned out to be biased against minority neighbourhoods; and US Justice Department risk-assessment systems, which have been proven to unfairly flag black defendants as future criminals. Other examples have been called out by the media: for example, the supply and demand pricing system that unexpectedly led to price gouging on airline tickets for Hurricane Irma evacuees. The New York Times reported at the time, "There are no ethics valves built into the system that prevent an airline from overcharging during a hurricane." [3]

In short, whether deliberately or unintentionally, there are now countless examples where software has not been designed to respect human values.

This article raises awareness about the importance of human values in software engineering. Based on insights from a number of real-world projects in the not-for-profit sector – which took place over a three year period – we make initial suggestions as to how human values can be handled in a software development process.

What are Human Values?

Human values have been studied extensively in social science, leading to several well-evidenced values theories. One of the most prominent is Schwartz's Theory of Universal Values (Figure 1). Based on surveys in 68 countries, Schwartz concluded that there are ten universal values, each subdivided into finer-grained values [4]. These values are universal in the sense that all countries and cultures recognize them as 'legitimate' values; however, an individual, organisation or society will hold a different subset of these values. Schwartz's model also makes no moral judgment. Thus, the value of 'wealth' is considered just as valid as the value of 'social justice'. Hence, there is a clear distinction between values and ethics: ethics are societally-agreed moral expectations, whereas values may be good or bad and have no moral connotation.

A related perspective on values is that of corporate values. Since the publication of the book, *Built to Last: Successful Habits of Visionary Companies* [5], which concluded that one key ingredient of success is a focus on corporate values, there has been a trend in the corporate world towards public expression of values. In a study by consultancy firm Maitland in 2015 [6], 83 of the FTSE100 companies articulate a clear set of corporate values. These values are typically social in nature and complement a corporation's need to drive profit margins.

Researchers in information systems and human-computer interaction have studied human values in design since the late 1980s. To date, however, the software engineering community has not leveraged this. Value-sensitive design [7] provides a suite of value-oriented techniques designed to elicit

stakeholder values, manage value conflicts, and influence high-level design decisions in software systems. These techniques are usually 'valuefied' versions of well-known methods: e.g., semi-structured interviews with stakeholders that focus on value elicitation, ethnographic methods viewed with a value lens, and focus groups used to discuss potential values concepts. Values in design [8] studies how social values are inherently embodied in technology and promotes a notion of 'engineering activism' whereby engineers are on the look out for designs that misalign with values and 'call them out'. Both value-sensitive design and values in design have a lot in common with participatory and co-design approaches to technology development, which emphasize a deep and thorough participation of users and other stakeholders in design decisions. These methods, however, come from a social science perspective and their application is therefore difficult or unclear in the technical world of software. Software engineering has a wide range of technical development methods (cf. design patterns, penetration testing, refactoring, software analytics); to date, the ideas and concepts from value-sensitive design and values in design have not been mapped down to the level of software techniques.

The word 'value' is unfortunately a rather overloaded term. Value-based software engineering [9] defines the notion of 'value' as largely economic. This is similar to the notion of value in some agile development methods, where the emphasis is on 'business value'.

Exploring Human Values in Practice

Our long term goal is to develop new software engineering practices that support practitioners in considering human values as part of software development. As a first step towards this goal, we undertook three real-world projects, each with customers from the not-for-profit sector. Two of these projects are referred to in this paper (Figure 2). In each case, we worked collaboratively with a team of developers, customers and other stakeholders to understand how values might be incorporated in the software development process.

Our research approach is exploratory – given that this problem has not been studied in depth before in a software engineering context, our intention was to experiment with adaptations of existing practices and generate insights that could be further validated at a later date.

Each project was chosen to be in a context where stakeholders had clear ideas about their values. In each project, a partnership was formed between researchers, software engineers and members of customer organisations and other stakeholders. Using a participatory agile development approach, each project developed an innovative software system in nine months that addressed a mutually identified social problem. The research team worked collaboratively with the organisations following participatory action research (PAR) principles.

For each project, we collected data on the development process and values intervention points, using a variety of methods, including semi-structured interviews, ethnographic observations, focus groups, co-design workshops, surveys and end-user trials.

Lessons Learned

Our initial studies suggested a number of ways in which human values can be incorporated into software development practice. These insights were derived from one or more of the methods applied, as

described above, including analysis of the data collected using standard quantitative and qualitative methods. We elaborate on four key insights.

Eliciting human values and making them more tangible

One of the immediate challenges when considering human values is the abstract nature of values and, in particular, how to make these more tangible in software design. Software engineers need concrete definitions to work with. It is likely impossible – and not desirable – to provide a universal definition for each value as, by their nature, values are very contextual. Instead, therefore, the task is to provide guidance on how each project can operationalize values definitions.

Based on our experience with the projects, we found it effective to use the Schwartz model as a starting point and then define relevant values using a technique we invented called *values portraits.*

The Schwartz model provides a clear taxonomy of human values to consider in a project. One starting point for a project, therefore, is to use the Schwartz model as a means to trigger discussion among project stakeholders as to what are the project values.

Once a subset of Schwartz values has been agreed upon, the challenge is to make these values more tangible in the context of the current project. We created *values portraits* for this purpose – see Table 1 for an example. A values portrait captures more detailed requirements as to what values mean in the specific context of a project. Thus, the value of 'reputation' can be refined into 'will not embarrass me' which can, in turn, be refined into concrete design decisions. We used structured text for values portraits, but the same approach could be applied using more formal modelling techniques.

A simple process for coming up with values portraits (which can be carried out collaboratively in a workshop-style or individually) is:

1. Discuss the values from the Schwartz model to see which best represent the project stakeholders' values. Note that, at this point, the project team may decide to use slightly different words to talk about their values – using the terms from the Schwartz taxonomy is less important than using words which resonate with project stakeholders.
2. Refine the agreed values using values portraits.
3. Extract functional and/or non-functional requirements from these portraits to complement requirements elicited elsewhere.

Values portraits were developed during the projects and then iteratively applied and proved effective. (See sidebar, "The Clasp Project", for an example.)

Documenting Values as the Rationale in Requirements

One key issue that came up is that for a values-driven approach to be successful, the project values must be reiterated throughout the project team at every opportunity. This is to ensure that consideration of values becomes part of the project culture.

We observed that values can provide the rationale for requirements. One view of requirements is that functional requirements capture the 'what' of a software system, whereas non-functional requirements capture the 'how'. We discovered that values capture the 'why' and thus provide one way of

augmenting traditional requirements with rationale as to how the requirements relate to broader human values.

This observation provides a way of documenting requirements and/or design decisions by providing values-based rationale where appropriate, which can help with reminding the team why certain decisions have been taken. (See sidebar, "The Clasp Project", for an example).

Considering Values throughout a Participatory, Agile Process

Once values have been documented within a project, they must be considered at all stages of the software development lifecycle. In our projects, we did not find the need for completely new development methods to support this. Rather, existing methods can readily be applied, but, crucially, they must be adapted.

We used a combination of participatory design and agile methods. Participatory design lends itself very well to considering values. Participatory design, however, often involves lengthy and heavy consultation with stakeholder groups as well as significant time spent co-designing solutions with participants. We found that typical participatory design methods are too slow for the fast-paced world of software, and, furthermore, that heavy levels of participation often do not lead to a better solution than a more lightweight approach [10]. In particular, some of the methods often used in participatory design, such as interviews and ethnography, can take a long time to produce results; we therefore adapted these methods to a 'quick and dirty' approach.

Similarly, agile methods lend themselves naturally to consideration of values: the inclusion of certain roles in an agile team (e.g., customer) provide natural points in which to insert values-driven thinking. Indeed, in some agile processes, certain team roles are intended to question and ensure that solutions are providing 'value'. However, 'value' in agile methods is solely concerned with economic value. One simple adaptation that we introduced was to assign a member of the agile team to the role of 'critical friend' to raise concerns about design decisions which may interfere with agreed values. This led to discussions around the appropriateness of design decisions and, in some cases, alternate decisions being taken.

Considering the whole, and values conflicts

One major insight from our projects is that a values-driven approach cannot simply consider software in isolation but must take a holistic approach. That is, even with careful design, there can be unintended consequences from introducing a software system, which may impact negatively on the values a project aims to uphold. Moreover, it is not always possible to satisfy all values within a project: values often inherently conflict, sometimes in a way that cannot be reconciled. In such cases, we took an approach that, where a particular value could not be fully satisfied within the software project, we would look for ways outside the project to partially address the value. The take-away message here is that, unsurprisingly, software cannot and should not attempt to be the sole enforcer of values; rather values need to be considered in a holistic, systems-thinking framework. (See sidebar, "The Patchworks Project", for an example).

Conclusion

We have argued in this article that human values should be considered as first-class citizens in software development. In the same way that quality concerns such as security and privacy have now become fully integrated in software development processes, so too should other human values such as transparency, respect, empowerment, and community responsibility. Indeed, the increasing prevalence of software in society means that now is an opportune time to consider human values in software.

In this article, we presented some early insights on how human values could be addressed in the not-for-profit sector. Further research is required to generalize these findings to other, more commercial industries. Our research group is also involved in efforts to integrate these insights into well accepted software methods such as agile development; hence, ongoing work is looking to formalize software processes for human values.

Like any software process, if values are to be handled properly, they must be considered in software specification, design and implementation, validation, and evolution. In essence, values-driven software engineering should include "valuefied" versions of well-known software engineering processes and techniques. Thus, for example, in requirements engineering, methods would allow business analysts to specify and refine a project's values, and, crucially, provide traceability to functional requirements. In software design, design patterns could be valuefied to make more explicit the potential impacts (positive and negative) on the project's values. Or there may be entirely new design patterns which capture best practice as to how certain values can be embedded in software. In implementation, developer tools could provide support to deal with values – this could be refactoring tools that check for breaking values, intelligent recommender systems that suggest the use of code fragments related to values in design patterns, or reverse engineering tools that would use automation to extract implicit values from code and make them explicit. In evolution, software engineering processes would be updated to allow developers to track values in software over time through (e.g.) new software metrics that give approximate measures of values. The challenge in all of this, of course, is the high level, abstract nature of values, but, as previously pointed out, if methods are provided to refine values into operational forms, technical software engineering solutions become viable.

Technical solutions for human values must go hand in hand with non-technical solutions. It may well be that new legal frameworks will emerge that will enforce a consideration of human values. And organisations will need to develop new structures for ensuring that staff are appropriately trained and supported in implementing human values. These socio-technical considerations should lead in turn to collaborative methods for dealing with values systematically: such as CERT-like repositories for documenting values breaches so future developers can avoid them, or values assurance cases, for organisations to provide evidence to regulatory authorities that they have properly considered human values.

In short, we believe that the next few decades will bring an increased awareness of human values in software development. As Grady Booch said in a 2015 keynote at the International Conference on Software Engineering, we must come to understand that "every line of code we write has a moral and ethical implication." Given the distinction between ethics and values, we prefer a different version of this statement – that "every line of code has a values implication" – but the sentiment is the same.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MS.2019.2956701, IEEE Software

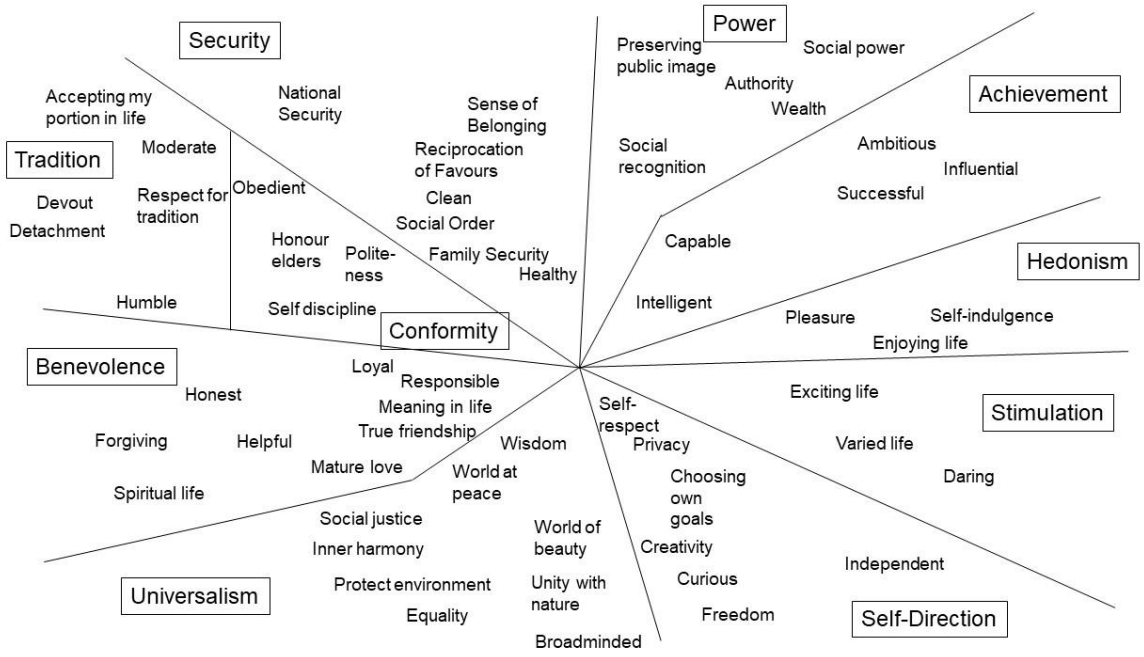

Figure 1: Theoretical Model of Relationships between Values in the Schwartz Theory of Basic Values (adapted from [11]). Boxed are the ten universal values, subdivided into finer-grained values. Values closer to each other are complementary; those further apart are in conflict.

(a) (b)

Figure 2: (a) Patchworks, a collaboratively designed information system for the homeless (http://vimeo.com/43110132); (b) Clasp [12], an anxiety tracking device for autistic adults (https://www.youtube.com/watch?v=WjvLZrEYE7M).

The Clasp Project (sidebar)

In the Clasp project (Figure 2), we worked with a group of around fifty autistic adults and their carers and family to develop new software that could tackle social anxiety. In particular, autistic people often use a form of self-stimulatory behaviour, known as stimming, to reduce their anxiety. Stimming is a repetitive movement, often involving objects (pulling on bracelets, repeatedly feeling soft materials, etc.). By 'digitizing' a person's stimming object, technology could provide a temporal map of levels of anxiety which could be triangulated with GPS location data, diary entries, and other sensor data to provide insights on what causes anxiety for a person and whether attempted behavioural interventions are helping. Traditional software qualities (privacy, performance, etc.) were important, but equally important were the human values of self-enhancement, openness to change, and conservation of reputation.

Table 1 shows an excerpt of the values portraits developed by the team. These include values expected of the technological solution and values expected of the process. These values were refined into requirements and influenced design decisions. For example, the value of Conservation, as expressed by "it won't embarrass me" related to fears of the autistic adults sharing information about their anxiety on social media. Participants wished to share information, but in a safe way. The decision was taken therefore to avoid well-known social networks in favour of the use of Diaspora, a social network whose stated principles (decentralization of data onto independently run servers, and freedom to use an invented identity) more closely aligned with Clasp's values.

We also documented the values in the statement of requirements – the values can be seen as the 'why' for a requirement. As an example, the project team decided that rather than only having sensing technology that passively collected data about users (e.g., GPS location, interactions with stimming objects), the system needed to also allow active interactions so the users would be in control. Hence, the requirement, "The system shall allow users to view and modify data collected" was augmented with, "Why? To support the value of feeling in control when reacting to stress." We found that these "why" statements were a good way to maintain a focus on values throughout the requirements and design process.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MS.2019.2956701, IEEE Software

Table 1: Values Portrait for Clasp.

Self-Enhancement The system gives me/other people opportunities to: • Harness my strengths • Better understand my behaviour • Address some of my difficulties • Feel more in control of my reactions to stress	Openness to Change The team will: • Be opportunistic with the unpredictable • Promote individual self-direction • Be 'un-disciplined' with methods • View technology as only part of the solution • Be open to other views
Conservation The system: • Won't embarrass me • Won't embarrass people around me • Won't cause nuisance to me and other people	Self-Transcendence The project will: • Enable team members to work in partnership with all stakeholders as peers • Support team-building by sharing tasks • Match primary responsibilities to individual skills

The Patchworks Project (sidebar)

In Patchworks (Figure 2), we worked with a group of homeless people and a local charity. The aim was to better understand the digital communication needs of the homeless, in a context where access to the internet is patchy and/or restricted. Participants quickly agreed that access to information (on health resources, temporary accommodation, etc.) with real-time updates (due to the constantly changing nature of such limited resources) was critical, but that current access (e.g., use of public computers) was not sufficient. The team therefore designed a bespoke RFID-based device that could be used to access personalized, real-time information from a set of 'stations' in public locations available 24 hours.

While traditional software qualities were important – data privacy, availability, low cost of devices, information accuracy – the Patchworks team was strongly inclined towards human values. Would the device empower the homeless? Would it lead to greater respect between and towards the homeless?

There were also clear tensions between values. The developers on the project had a strong desire for novelty and innovation. However, the charity needed something much more mundane (e.g., a new website). These values of novelty and competitiveness, then, contrasted with the need for trust, respect and empowerment that the homeless participants were hoping for.

The team needed to develop ways of dealing with these values tensions. The team therefore developed ways to provide more 'mundane' benefits at the same time as exploring more innovative solutions. For example, the charity needed a new website: whilst developing a website was out of scope for the project, the team managed to facilitate a connection with a local web developer. This is an example where the team found a mutually beneficial solution satisfying two sets of competing values but in a way that advanced the project overall.

References

[1] Bojan Georgievski & Anas Alqudah (2016). The Effect of the Volkswagen Scandal (A Comparative Case Study), Research Journal of Finance and Accounting, 7(2), 54-57.

[2] Sainyam Galhotra, Yuriy Brun & Alexandra Meliou (2017). Fairness testing: testing software for discrimination, Proceedings of ESEC/SIGSOFT FSE 2017, 498-510.

[3] Justin Sablich (2017). 'Price Gouging' and Hurricane Irma: What Happened and What to Do, New York Times, Sep 17. Retrieved from https://www.nytimes.com/2017/09/17/travel/price-gouging-hurricane-irma-airlines.html.

[4] Shalom Schwartz (2012). An Overview of the Schwartz Theory of Basic Values, Online Readings in Psychology and Culture, 2(1).

[5] James Collins & Jerry Porras (1997). Built to last: successful habits of visionary companies, New York: HarperBusiness.

[6] Simon Walker, Foreward (2015). The values most valued by UK plc, Retrieved from http://www.maitland.co.uk/wp-content/uploads/2015/10/20151001-Maitland-Values-Report.pdf

[7] Batya Friedman, David Hendry & Alan Borning (2017). A survey of Value Sensitive Design methods. Foundations and Trends in Human Computer Interaction. Boston and Delft: Now Publishers.

[8] Cory Knobel and Geoffrey Bowker (2011). Values in Design, Communications of the ACM, 54(7), 26-28.

[9] Stefan Biffl, Aybuke Aurum, Barry Boehm, Hakan Erdogmus & Paul Grunbacher (2006). Value-based software engineering, Springer.

[10] Jon Whittle (2014). How much participation is enough?: a comparison of six participatory design projects in terms of outcomes. Participatory Design Conference, 121-130.

[11] Shalom Schwartz (2006). Basic human values: Theory, measurement, and applications. Revue Francaise de Sociologie, 47, 929-985.

[12] Maria Angela Ferrario, William Simm, Stephen Forshaw, Adrian Gradinar, Marcia Tavares Smith & Ian C. Smith (2016). Values-first SE: research principles in practice. Proceedings of ICSE (Companion Volume), 553-562.