Responsible Software Engineering: Requirements and Goals

Amel Bennaceur, Carlo Ghezzi, Jeff Kramer, and Bashar Nuseibeh

Abstract In this chapter, we provide an introduction to the discipline of requirements engineering as part of the software engineering process. We indicate how to elicit, articulate, and organize the goals of complex software systems as an explicit expression of the requirements that the proposed or existing software system is expected to achieve and maintain, including what the system should avoid performing. We advocate that system requirements goals can and should be used to explicitly capture, express, and reason about the diverse digital humanism values which are of concern in socio-technical systems. This is an essential aspect of responsible software engineering.

1 Introduction

Software is creating a new digital world in which humans live, individually and socially. This is a large and complex *socio-technical system* where the boundaries between digital, physical, and social spaces are increasingly disappearing. Many activities in such a system are automated, supporting and sometimes replacing human work and creating new functionalities that did not exist before. Humans interact with software-enabled agents in their daily life. Software now defines and administers most of the laws that govern the world. This was observed in the late 1990s by Lawrence Lessig, in his framing of "Code is Law" (Lessig, 2000).

A. Bennaceur (✉) · B. Nuseibeh
The Open University, Milton Keynes, UK

Lero, University of Limerick, Limerick, Ireland
e-mail: amel.bennaceur@open.ac.uk; bashar.nuseibeh@open.ac.uk

C. Ghezzi
DEIB, Politecnico di Milano, Milan, Italy
e-mail: carlo.ghezzi@polimi.it

J. Kramer
Imperial College London, London, UK
e-mail: j.kramer@imperial.ac.uk

H. Werthner et al. (eds.), *Introduction to Digital Humanism*,
https://doi.org/10.1007/978-3-031-45304-5_20

Software engineers, who create code, are the *demiurges*. Although they are responsible for "technical" decisions, the consequences of their decisions go far beyond the purely technical sphere, often with unintended and unanticipated consequences. At the same time, legal systems have lagged behind in adapting to technological changes.

How can the implications of socio-technical systems developed by software engineers be properly considered when systems are conceived and developed? How can the values and issues of digital humanism drive the software engineering process? How does that engineering process interact with other processes (political, normative, etc.) both ex ante—while the system is designed—and ex post, when systems are deployed and operate?

Engineers have traditionally focused on functional correctness, efficiency, and scalability of their solutions. By and large, they have ignored fairness, inclusivity, and deep consideration of the social implications of their solutions. They have mastered technology and make complex technical decisions, but rarely consider the consequences of future use and misuse of their products in society.

In this chapter, we advocate that these issues and values must be considered and explicitly integrated into the software engineering process: in particular, in the explicit expression of the requirements that the proposed or existing system is expected to achieve and maintain. This focus on so-called requirements engineering can provide a bridge between the world in which digital humanism values arise and the digital machine that software engineers design, build, and deploy in that world.

We begin with an overview of requirements engineering, focusing on ways in which goals and requirements are elicited from diverse stakeholders and how they can be explicitly modeled and analyzed. We illustrate how such goal models can be extended to capture various human values and discuss how they can be analyzed for the purpose of validation and verification. We conclude with a discussion on a more responsible software engineering discipline and some suggested exercises to engage students in the articulation of and reflection on digital humanism goals in software systems.

2 Requirements Engineering (RE)

RE has been the subject of several popular books and surveys; this section gives a brief introduction to requirements as a primary basis for sound software engineering. It also provides relevant references for further exploration of the area.

2.1 Introduction to RE

> *Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.* (Zave, 1997)

This definition by Zave emphasizes that a new software system is introduced to solve a real-world problem and that a good understanding of the problem and the associated context is at the heart of RE. Therefore, it is important not only to define the *goals* of the software system but also to *specify its behavior* and to understand the *constraints and the environment* in which this software system will operate. The definition also highlights the need to consider change, which is inherent in any real-world situation. Finally, the definition suggests that RE aims to capture and distill the experience of software development across a wide range of applications and projects.

Although Zave's definition identifies some of the key challenges in RE, the nature of RE itself has been changing. First, RE is not specific to software alone but to *socio-technical systems* in general, of which software is only a part. Software today permeates every aspect of our lives, and therefore, one must not only consider the technical but also the physical, economical, and social aspects. Second, an important concept in RE is *stakeholders*, that is, individuals or organizations who stand to gain or lose from the success or failure of the system to be constructed (Nuseibeh & Easterbrook, 2000). Stakeholders play an important role in eliciting requirements as well as in validating them.

While the definition of the requirements helps delimit the solution space, the requirement problem space is less constrained, making it difficult to define the environment boundary, negotiate the resolution of conflicts, and set acceptance criteria (Cheng & Atlee, 2007). Therefore, several guidelines are given to define and regulate the RE processes in order to build adequate requirements (Robertson & Robertson, 2012). Figure 1 summarizes the main activities of RE:

Elicitation. Requirements elicitation aims to discover the needs of stakeholders as well as understand the context in which the system-to-be will operate. It may also explore alternative ways in which the new system could be specified. Several techniques can be used including (i) traditional data gathering techniques (e.g., interviews, questionnaires, surveys, analysis of existing documentation), (ii) collaborative techniques (e.g., brainstorming, workshops, prototyping), (iii) cognitive techniques (e.g., protocol analysis, card sorting), (iv) contextual techniques (e.g., ethnographic techniques, discourse analysis), and (v) creativity techniques (e.g., creativity workshops, facilitated analogical reasoning).

Modeling. The results of the elicitation activity often need to be described precisely and in a way accessible by domain experts, developers, and other stakeholders. A wide range of techniques and notations can be used to represent requirements, ranging from informal to semi-formal to formal (mathematical) methods. The choice of the appropriate method often depends on the kind of analysis or reasoning that needs to be performed.

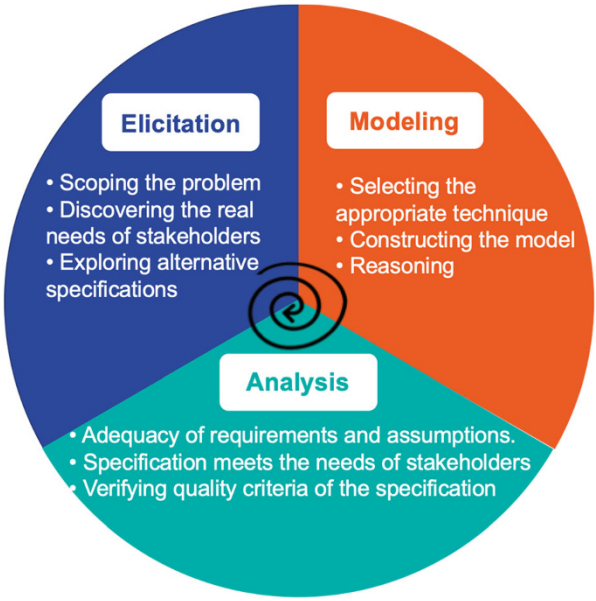

Fig. 1 Main activities of requirements engineering

Analysis and Assurance. Requirements quality assurance seeks to identify, report, analyze, and fix defects in requirements. It involves both validation and verification. Validation aims to check the adequacy of the modeled requirements and domain assumptions with the actual expectations of stakeholders. Verification covers a wide range of checks including quality criteria of the modeled requirements (e.g., consistency).

2.2 Requirements and Goals

Zave and Jackson (1997) suggest that there are three main kinds of artifacts that requirements engineers would produce during the RE activities:

- Statements about the *domain*, describing properties that are true regardless of the presence or actions of the machine (or software system)
- Statements about *requirements*, describing properties that the stakeholders want to be true of the world in the presence of the machine
- Statements about the *specification*, describing what the machine needs to do to achieve the requirements

These statements can be written in natural language, formal logic, semi-formal languages, or indeed some combination of them, and Zave and Jackson are not prescriptive about that. What is important is their relationship: *The specification of the machine, together with the properties of the domain, should satisfy the requirements.*

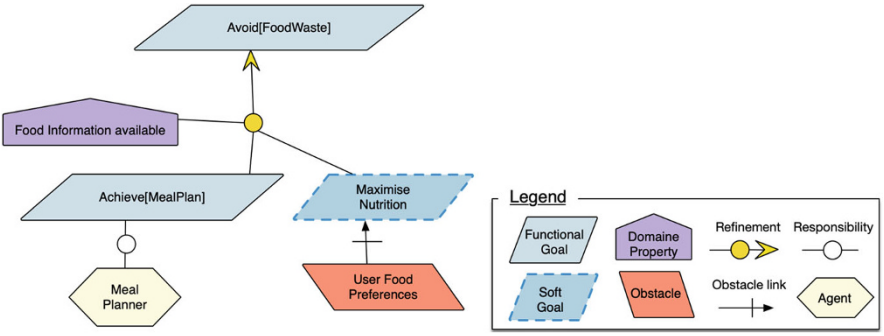

Fig. 2 A sample goal model for the MealPlanning example

To illustrate those notions, let us consider an example of *meal planning* as a way to tackle an important and pressing societal problem, food waste. It is estimated that food waste per capita by consumers in Europe and North America is 95–115 kg/year. Food waste is often caused by insufficient planning of purchases and consumption by individuals. Effective strategies to reduce wasteful behavior should require minimum time and cognitive effort from consumers. The Feed me, Feed me exemplar (Bennaceur et al., 2016) describes a system based on the Internet of Things to support the production, distribution, and consumption of food. We use ideas and challenges from the Feed me, Feed me exemplar to focus on how our approach can support individuals in reducing food waste in households.

For example, to avoid food waste, we should plan meals. This can be achieved by refinements (as illustrated in Fig. 2). The *Avoid[FoodWaste]* goal is refined into sub-goals and associated domain properties. A *goal* in this model is defined as a prescriptive statement that the system should satisfy through the cooperation of *agents* such as humans, devices, and software. Goals may refer to services to be provided (functional goals) or quality of service (soft goals).

Achieve[MealPlan] is a functional goal, while *MaximiseNutrition* is a soft goal. While functional goals can be satisfied or not, soft goals are often optimized. Keywords such as *Achieve*, *Maintain*, and *Avoid* are used to characterize the intended behaviors of the goals and can guide their formal specification.

Domain properties are descriptive statements about the environment. For example, *Food Information Available* is a domain property. An important relationship is that the goal *Avoid[FoodWaste]* can be satisfied through *Achieve[MealPlan]* and *MaximiseNutrition* assuming *Food Information Available.*

Besides describing the contribution of sub-goals (and associated domain properties) to the satisfaction of a goal, refinement links are also used for the operationalization of goals and assigning them to (software) agents. For example, *MealPlanner* is responsible for satisfying the goal *Achieve[MealPlan].*

Finally, *Conflict* links are used to represent obstacles to the satisfaction of goals. For example, *UserFoodPreferences* may hinder the satisfaction of *MaximiseNutrition*. Reasoning about obstacles enables risk analysis of the goals by eliciting properties that may obstruct the satisfaction of goals.

Hence, RE is grounded in the real world; it involves understanding the *environment (domain)* in which the system-to-be will operate and defining a detailed, consistent *specification* of the software system-to-be. This process is incremental and iterative as illustrated in Fig. 2.

Zave and Jackson specify five clear criteria for this process to complete:

- Each goal has been validated with the stakeholders.
- Each domain property has also been validated with the stakeholders.
- The goal does not constrain the environment or refer to the future.
- There exists a proof of the satisfaction of goals.
- The goals and domain properties are consistent.

2.3 The Need for Human-Centered Values

The essence of RE is a good understanding of problems, which includes analyzing the domain, communicating with stakeholders, and preparing for system evolution. However, techniques such as machine learning, automated compositions and interactions, and creativity disrupt the traditional models of software development and call for quicker, if not immediate, response from requirements engineering. Moreover, the social underpinning and the increasing reliance on software systems for every aspect of our life call for better methods to understand the impact and implications of software solutions on individuals and society as a whole.

For example, several pressing global problems such as climate change and sustainability engineering as well as increasingly important domains such as user-centered computing and other inter- and cross-disciplinary problems challenge existing processes and techniques. It is no longer enough to understand the needs of stakeholders and the constraints of the environments in which a software system is deployed; we also need to understand the values of the stakeholders and understand the broader impact of deploying software solutions. In the next section, we move to values and their interaction with requirements.

3 Values We Live By: Eliciting, Articulating, and Organizing Goals

Digital humanism argues for adopting a broader framework where, besides the technical perspective, multiple perspectives (including ethical, social, legal, political, and economic) are considered when developing systems that have an impact on individuals and society.

Recent work has promoted the need to consider ethics and values during the development of software systems (Whittle, 2019). As outlined by Mougouei et al.

(2018), "people are demanding that practitioners align technologies with human values." Some approaches have been proposed to assess and study values in software engineering (Winter et al., 2019), to incorporate social values in software design patterns (Hussain et al., 2018), and to measure the impact of values in requirements engineering activities (Perera et al., 2021). Values are well studied in human-computer interaction and information systems (Cockton, 2004).

For RE, this means rethinking the world in terms of broader and changing stakeholders, their needs, and their values. It also means rethinking the notion of requirements satisfaction to incorporate values and the inevitability of failure and change. Some of the challenges of doing so stems from the subjectivity and uncertainty of values. Values are *subjective* and depend on the diverse viewpoints of stakeholders because different stakeholders describe value requirements differently. As a result, they have different and sometimes contradictory requirements. For example, if we consider the value of fairness, serving a protected group with priority can promote fairness in society, but, at the same time, it may seem discriminatory to others. Values are *uncertain* and are often better understood once the software solution is deployed. For example, awareness of gender bias in data may lead to the deployment of existing equality policies, and their impact and consequences are better understood once deployed.

The debate has long focused on principles and codes of conduct for considering values in software systems. However, it is increasingly moving to tools and processes for implementing those values and principles in practice. While awareness of the potential issues is increasing at a fast rate, the software/requirements engineering community's ability to take action to mitigate the associated risks is still in its infancy. There is still a need to close the gap between principles and practices for engineers to apply ethics at each stage of the development pipeline and to signal to researchers where further work is needed. In other words, we need methods to move from "what" values to embed to "how" those values can be embedded in software systems. This section provides some direction toward achieving this goal.

3.1 Values and RE Activities

Let us first review the RE activities with humanistic values in mind.

Elicitation. Social scientists, ethicists, philosophers, policymakers, technologists, and civil society have been involved in a debate around what is necessary to enable society to capitalize on the opportunities of software systems while ensuring fair and ethical decision-making is maintained. Participatory design aims to elicit the values of multiple stakeholders by following several steps, which include:

- Involving actual users for eliciting value concerns
- Using personas to consider/assume user values
- Using prototypes to analyze assumptions about values
- Using diversity in members selected from various stakeholder groups
- Focusing on cultural sensitivities
- Being considerate of language needs of different stakeholder groups
- Developing empathy with users, emulating their experiences
- Building an atmosphere of trust for stakeholders to voice their opinions
- Applying user feedback to improve mock-ups and prototypes

In addition to continual engagement with stakeholders and practitioners, reflection on practices and the impact of the developing software systems is equally important. The Self-Reflection Tool of the Responsible Research and Innovation (RRI)[1] framework helps practitioners consider the societal and ethical issues that may be involved with technology. Learning by doing underpins the AREA (Anticipate, Reflect, Engage, and Act) approach to RRI. This means that professional and social responsibility is best developed through experience and reflective practice. The guidelines for such practices include:

1. Involving a wide range of actors and people in practice, deliberation, and decision-making. This strengthens democracy and broadens sources of expertise, disciplines, and perspectives.
2. Envisioning impact and reflecting on the underlying assumptions, values, and purposes to better understand how the developed systems shape the future. This yields valuable insights and increases the capacity to act on what we know.
3. Communicating in a meaningful way the methods, results, conclusions, and implications to enable public scrutiny and dialogue. This benefits the visibility and understanding of the developed systems.
4. Being able to modify modes of thought and behavior, overarching organizational structures, in response to changing circumstances, knowledge, and perspectives. This aligns action with the needs expressed by different stakeholders.

Modeling. In Value-Based Requirements Engineering (Thew & Sutcliffe, 2018), values are seen as personal attitudes and beliefs which influence functional and non-functional requirements. There is evidence of human values being treated as software requirements, specifically as soft goals or non-functional requirements (Barn, 2016). In values-first software engineering, Ferrario et al. (2016) argue that complex wicked problems such as sustainability should be treated as soft goals, not as functional requirements. Nurwidyantoro et al. (2022) postulate that non-functional requirements can be seen as a subset of human values and propose to classify human values and align them to system values. They found system value themes, such as efficiency and usability, similar to non-functional requirements.

[1] https://rri-tools.eu/. Accessed 10 April 2023.

Fig. 3 Dimensions to consider for eliciting and operationalizing values

Assurance. Operationalizing values is defined as "the process of identifying human values and translating them to accessible and concrete concepts so that they can be implemented, validated, verified, and measured in software" (Shahin et al., 2022). It is common for stakeholders to gain a better understanding of their values as they experience, reflect, and learn more about them (Gentile, 2010). However, elicitation and modeling approaches focus on early stages of the development process, with little attention given to the satisfaction of values in deployed software systems (Shahin et al., 2022). Software solutions can help stakeholders articulate, measure, and reflect on their values while they are experiencing the software. Values@Runtime (Bennaceur et al., 2023) deal with uncertainty by delaying some decisions until software is in operation. It adopts an adaptive process to engage stakeholders and to support learning about models of stakeholders' values. It provides values instantiation as a means of representing the concrete actions that stakeholders associate with values (Hanel et al., 2017). This framework supports values operationalization in terms of (i) representation, instantiation, and monitoring of values and behavior; (ii) understanding existing mismatches between values and users' behavior based on analysis; and (iii) recommending ways to align values and behavior as well as reflecting on the recommendations.

Hence, eliciting and operationalizing values involves three dimensions (see Fig. 3):

- People, through the adoption of a human-centered view and participatory design as well as involving a diversity of stakeholders and teams
- Artifacts, by making explicit value statements and engineering systems for diverse stakeholders
- Processes, by linking values between requirements and implemented software and by being transparent and open to accountability about implementation practices and mindful of project impact and following current standards and regulations

3.2 Values and Goals

Let us consider the example of fairness when food shopping (Farahani et al., 2021). The high-level goal is *Achieve[FairShopping]* and might be refined in multiple ways—see Fig. 4.

For example, when the domain property *AbundantStock* holds, then the goal is to *maximize Products Sold*, which can be operationalized by allowing users to buy as many products as they want/need. When the domain property Limited Stock holds, then there needs to be a choice between two goals: *Achieve[EquitableAccess]* by prioritizing protected groups or *Achieve[EqualAccess]* by limiting the maximum amount of product per shopper without distinction between shoppers. While not mutually exclusive, the choice is driven by consideration of multiple stakeholders, e.g., supermarkets' willingness to implement different procedures, government's willingness to support protected groups, and public acceptance of prioritizing protected groups. For example, prioritizing a protected group can be perceived as fair for some people, but at the same time, it may seem discriminatory to others. In other words, a goal model can help highlight the stakeholders involved when making value-sensitive choices, e.g., fair for whom or who is responsible for the choice. The goal model helps highlight and contrast alternative operationalization of values.

Emotions can be used as proxy to values and leveraged to design inclusive processes (Hassett et al., 2023). For example, the *Supermarket* might want the stakeholder group, *Vulnerable Shopper* (e.g., older person or person with special needs), to *feel Cared for*, which then leads to prioritizing protected groups.

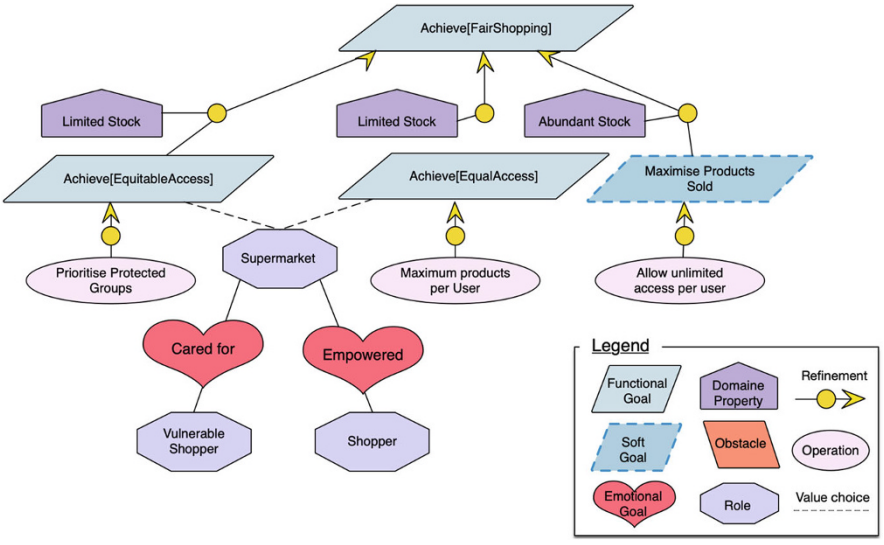

Fig. 4 A sample (emotional) goal model for the fair food shopping example

4 Toward Responsible Software Engineering: DigHum Goals in the Life Cycle

In this section, we discuss how the principles of digital humanism may guide the life cycle of socio-technical systems: from the conception and development of a system, to its operation in the real world, to its continuous evolution.

4.1 Requirements and Other Activities

The first and most important step consists of understanding and articulating the requirements. The previous two sections shed light on this crucial activity, through which developers assume explicit responsibility with respect to the system under development. Through goal models, they express a contract with stakeholders and future users, which states what the system is expected to achieve. Traditionally, software engineers are educated to focus on goals that refer to the functionalities and expected behaviors to be provided by the system and on technical qualities, like efficiency (e.g., average response time of certain transactions), portability of the implementation on different architectures, or security (e.g., guaranteed restricted access to certain data or functionalities). In our context, however, requirements also reflect the general humanistic values, modeled as explicit goals to be met by the future system. For example, fairness is explicitly modeled as a goal to achieve in the context of the food shopping example.

Eliciting and articulating these goals is critical, but also quite difficult and highly context dependent. The technical skills possessed by software engineers alone may fall short. Not only stakeholders and user representatives must be involved, but also experts from other domains—like philosophy (ethics), history, social sciences, economics, or law—may have a lot to say in order to understand goals, analyze and resolve conflicts, and prioritize among them, but also to anticipate possible uses (or misuses) of socio-technical systems in the real world. Depending on the specific system being developed, a deliberative process needs to be put in place, which gives voice to different viewpoints and then responsibly leads to decisions that inform all subsequent development steps.

Requirements are a prerequisite for design and implementation (Fig. 5). These are technical steps that lead to a functioning socio-technical system. Design is responsible for defining the software architecture, i.e., decomposing a system into components and deciding how different components interact and communicate. Implementation is responsible for producing an executable system, often through a combination of programmed parts, libraries, and software frameworks.

However, requirements also permeate many parts of the systems development process. During system design, requirements are used to inform decision-making about different design alternatives. During system implementation, requirements are used to enable system prototyping and testing. Once the system has been deployed,

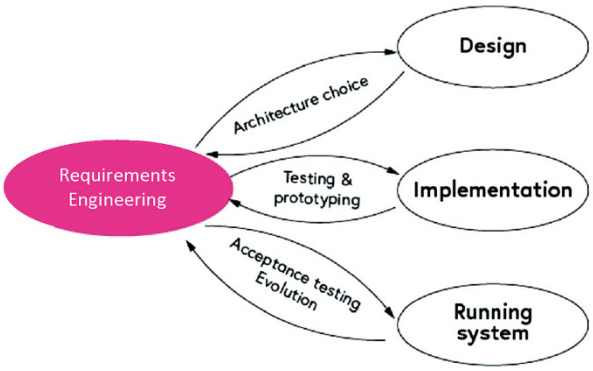

Fig. 5 Crucial relationship between requirements and other activities

requirements are used to drive acceptance tests to check whether the final system does what the stakeholders originally wanted. In addition, requirements are reviewed and updated during the software development process as additional knowledge is acquired and stakeholders' needs are better understood.

Each step of the development process may lead to the definition of additional requirements through a better understanding of the domain and associated constraints. Therefore, there is a need to consider requirements, design, and architecture concurrently, and this is often the process adopted by software engineers.

4.2 Software Processes

Different process models can be followed to guide development, ranging from top-down (waterfall) processes to bottom-up and iterative processes. Waterfall processes are monolithic and sequential: they try to strictly enforce completion of the requirements phase before proceeding to the design phase, which must itself be completed before moving to implementation. Strict sequential ordering of phases is only suitable for highly structured systems that operate in well-defined, formalizable, and highly stable contexts. It is not suitable for ill-defined and unstable settings as is the case in most socio-technical systems, where humans play a fundamental role. More flexible—iterative and incremental—life cycles, such as the popular process models which fall under the term *agile processes*, are almost always adopted for the latter kind of systems. Agile processes, which envision the development of system increments, e.g., via *sprints* in the SCRUM agile methodology (Schwaber & Beedle, 2002), appear as a suitable setting to accommodate the necessary deliberations through which digital humanism-inspired requirements can be explored and then guide development. The chapter by Zuber et al. in this book provides deeper insights into how agile development methods are inherently suitable for embedding digital humanism values into software systems.

4.3 Validation and Verification

Two other important activities need to be carried out during development: verification and validation (V&V). The two terms shed light on two complementary kinds of assurances. *Validation* is the assurance that the system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. *Verification* is the assurance that the system complies with its specification. The two terms would of course be synonyms if specifications were exhaustive and complete. This is almost inevitably impossible in practice. In addition, as we discuss next, the needs of customers and other stakeholders continuously evolve, and therefore, an upfront complete specification is impossible to realize.

V&V is itself not a stage of system development, but rather a cross-cutting activity that permeates all development steps. Requirements are continuously verified and validated as they are elicited and formalized; likewise, architectures and implementation increments are subject to V&V. Delivery of (partial) applications for real use presupposes an adequate level of V&V to check compliance with specifications, including possible existing regulations, and adherence to users' needs. It is also possible to design systems in a way that these checks are made automatically by the system while it operates, at run-time (run-time V&V).

V&V is practiced through two complementary approaches: systematic reasoning and testing. Systematic reasoning tries to analyze the artifacts under development to prove that the stakeholders' expectations are met by affirming that violations of those expectations are impossible. Testing develops experiments that try to bring the system into desirable and undesirable states, to collect empirical evidence that the system being developed can be delivered for practical use. The two approaches are complementary, since exhaustive testing is impossible to achieve and tools to assist in systematic reasoning do not scale up to large systems.

4.4 The Running System

The life cycle of an application does not end when it is deployed. Most systems, and especially those successfully used in practice, are subject to continuous evolution, traditionally called maintenance. New requirements may arise from real use, pre-existing requirements may need to be adapted due to new insights gained while the system has been in use, opportunities for improvements may be discovered, and errors or other problematic situations that evaded V&V may show up during execution. To support evolution, specifically designed monitors may be implemented in the deployed applications to perform run-time V&V, checking for the insurgence of potential risks, violations of desirable policies, or mishaps.

5 Conclusions

In this chapter, we have explained the central role of requirements engineering in the software engineering process of software production and evolution. We have explained why we advocate requirements specifications and goals as the most promising and pragmatic technique to explicitly express the societal and digital humanism values which are so crucial to sound and responsible software engineering of socio-technical systems. This includes not just what the system should achieve but also what it should avoid performing. Some form of continuous monitoring of the running system will be needed to support assessment as part of responsible software engineering. This will also require that software engineers are involved in outreach activities regarding the global effects of their products: to assess impact, use, and abuse. As mentioned, experts in other disciplines (social scientists, lawyers, etc.) will also need to be involved, not just at the elicitation stage but also when the system is deployed and running. This diversity of stakeholders is becoming more and more important as systems are embedded in society.

We believe that this extension of traditional software engineering to include humanistic values is essential to cope with complex socio-technical systems. It will inevitably require further research, practice, and education to refine the techniques, to gain further empirical evidence and experience, and to ensure dissemination to the profession.

Discussion Questions for Students and Their Teachers
In the following hypothetical projects work together with colleagues from different disciplines and with different backgrounds, to articulate the Digital Humanism goals and overall requirements to be reached by a hypothetical socio-technical system, understanding potential conflicts, and mitigating potential risks, including misuse.

Consideration should also be given to what should be automated and what is left to humans to perform and also whether it is possible to ascertain whether or not the resulting system is compliant with the specified goals and explicit values.

1. Hypothetical Project 1: Citizen Forensics

   The police are overstretched, criminality is on the rise, . . . how can citizens participate in deterring crime and helping the police (and each other) detect anti-social incidents and solve crime.

   Hints/issues: you could explore risk and issues around surveillance (before or after incidents), harassment, privacy, citizen-police relations, and information sharing. . .A resource: https://www.citizenforensics.org
2. Hypothetical Project 2: Technology in the Courtroom

   It's not easy to be a judge. . . It is necessary to assess as correctly as possible whether an offender will recidivate, what sentence is appropriate for the particular offense, whether or not the sentence should be suspended, and much more.

   Wouldn't it be great if technology could make judgments easier?

   Many ethical questions arise here, such as who is "fairer," a technical system or a human? How transparent must a decision by a technical system be that

supports a court ruling? Is such a system more of a science fiction fantasy à la Minority Report or an actual chance to counter prejudices, perception biases, or even racist tendencies among judges?

Here are some papers and articles about this topic:
https://link.springer.com/content/pdf/10.1007/s10506-022-09310-1.pdf
https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm
https://link.springer.com/article/10.1007/s10506-022-09312-z
https://scholarship.law.umn.edu/mlr/58/

Proposed by: Anna Dhungel

3. Hypothetical Project 3: My Truth, Your Truth

Since the corona pandemic, I don't recognize some of my friends. Through some social media forums, they have become vaccination opponents, mask deniers, and world conspirators.... How can social media be made social and responsible again without immediately giving the feeling of living in a "dictatorship of opinion"?

Hint: The following article gives a brief introduction into the democratic roles of news recommender systems: https://doi.org/10.1080/21670811.2019.1623700.

Proposed by: Kian Schmalenbach and Eva Gengler

Learning Resources for Students

1. Van Lamsweerde, A., 2009. Requirements engineering: From system goals to UML models to software (Vol. 10, p. 34). Chichester, UK: John Wiley & Sons.

The book presents a systematic method to elaborate complex system models, analyze them, and derive software specifications from them. The method is known as KAOS (Keep All Objectives Satisfied). The goal models in this chapter used notations and formalisms from this book.

2. Brey, P. and Dainow, B., 2021. Ethics by design and ethics of use in AI and robotics. The SIENNA project-Stakeholder-informed ethics for new technologies with high socioeconomic and human rights impact.

The document provides guidance for including ethical principles and procedures into the design and development processes of AI systems.

3. IEEE Standard Model Process for Addressing Ethical Concerns during System Design, in IEEE Std 7000–2021, vol., no., pp.1–82, 15 Sept. 2021, doi: https://doi.org/10.1109/IEEESTD.2021.9536679.

The standard establishes a set of processes by which engineers and technologists can include consideration of ethical values in system design and development.

4. Guszcza, J., Danks, D., Fox, C., Hammond, K., Ho, D., Imas, A., Landay, J., Levi, Ma., Logg, J., Picard, R., Raghavan, M., Stanger, A., Ugolnik, Z., Woolley, A., Hybrid Intelligence: A Paradigm for More Responsible Practice (October 12, 2022). Available at SSRN: https://ssrn.com/abstract=4301478 or https://doi.org/10.2139/ssrn.4301478.

The paper presents the hybrid intelligence paradigm, aimed at supporting a more responsible practice, through simultaneous consideration of machine capabilities and human psychology, behaviors, needs, and values in the development of AI-based systems.

Acknowledgments This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/V026747/1 and EP/R013144/1] and Science Foundation Ireland [grant number 13/RC/2094\P2].

References

Barn, B. S. (2016). Do you own a Volkswagen? Values as non-functional requirements. In Human-centered and error-resilient systems development: IFIP WG 13.2/13.5 joint working conference, 6th international conference on human-centered software engineering, HCSE 2016, and 8th international conference on human error, safety, and system development, HESSD 2016, Stockholm, Sweden, August 29-31, 2016, Proceedings 8 (pp. 151–162). Springer.

Bennaceur, A., McCormick, C., Galán, J. G., Perera, C., Smith, A., Zisman, A., & Nuseibeh, B. (2016). Feed me, feed me: An exemplar for engineering adaptive software. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems* (pp. 89–95).

Bennaceur, A., Hassett, D., Nuseibeh, B., & Zisman, A. (2023). Values@ runtime: An adaptive framework for operationalising values. In *Proceedings of the 45th IEEE/ACM international conference on software engineering—software engineering in society track*.

Cheng, B. H., & Atlee, J. M. (2007). Research directions in requirements engineering. Future of software engineering (FOSE'07), (pp. 285–303).

Cockton, G. (2004). Value-centred HCI. In *Proceedings of the third Nordic conference on human-computer interaction* (pp. 149–160).

Gentile, M. C. (2010). *Giving voice to values: How to speak your mind when you know what's right*. Yale University Press.

Farahani, A., Pasquale, L., Bennaceur, A., Welsh, T., & Nuseibeh, B. (2021). On adaptive fairness in software systems. In 2021 International symposium on software engineering for adaptive and self-managing systems (SEAMS) (pp. 97–103). IEEE.

Ferrario, M. A., Simm, W., Forshaw, S., Gradinar, A., Smith, M. T., & Smith, I., (2016). Values-first SE: Research principles in practice. In *Proceedings of the 38th international conference on software engineering companion* (pp. 553–562).

Hanel, P. H., Vione, K. C., Hahn, U., & Maio, G. R. (2017). Value instantiations: The missing link between values and behavior?. Values and behavior: Taking a cross cultural perspective (pp. 175–190).

Hassett, D., Bennaceur, A., & Nuseibeh, B. (2023). Feel it, code it: Emotional goal modelling for gender-inclusive design. In *Requirements engineering: Foundation for Software Quality: 29th international working conference, REFSQ 2023, Barcelona, Spain, April 17–20, 2023, proceedings* (pp. 324–336). Springer Nature.

Hussain, W., Mougouei, D., & Whittle, J. (2018). Integrating social values into software design patterns. In *Proceedings of the international workshop on software fairness* (pp. 8–14).

Lessig, L., (2000). Code is Law. Harvard Magazine. https://www.harvardmagazine.com/2000/01/code-is-law-html.

Mougouei, D., Perera, H., Hussain, W., Shams, R., & Whittle, J. (2018). Operationalizing human values in software: A research roadmap. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering* (pp. 780–784).

Nurwidyantoro, A., Shahin, M., Chaudron, M. R., Hussain, W., Shams, R., Perera, H., Oliver, G., & Whittle, J. (2022). Human values in software development artefacts: A case study on issue discussions in three android applications. *Information and Software Technology, 141*, 106731.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. In *Proceedings of the conference on the future of software engineering* (pp. 35–46).

Perera, H., Hoda, R., Shams, R. A., Nurwidyantoro, A., Shahin, M., Hussain, W., & Whittle, J. (2021). The impact of considering human values during requirements engineering activities. *arXiv Preprint*. 2111.15293.

Robertson, S., & Robertson, J. (2012). *Mastering the requirements process: Getting requirements right*. Addison-Wesley.

Shahin, M., Hussain, W., Nurwidyantoro, A., Perera, H., Shams, R., Grundy, J., & Whittle, J. (2022). Operationalizing human values in software engineering: A survey. *IEEE Access, 10*, 75269–75295.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Prentice Hall.

Thew, S., & Sutcliffe, A. (2018). Value-based requirements engineering: Method and experience. *Requirements Engineering, 23*, 443–464.

Winter, E., Forshaw, S., Hunt, L., & Ferrario, M. A. (2019). Advancing the study of human values in software engineering. In *2019 IEEE/ACM 12th international workshop on cooperative and human aspects of software engineering (CHASE)* (pp. 19–26). IEEE.

Whittle, J. (2019). Is your software valueless? *IEEE Software, 36*(3), 112–115.

Zave, P. (1997). Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR), 29*(4), 315–321.

Zave, P., & Jackson, M. (1997). Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology (TOSEM), 6*(1), 1–30.