9 Epilogue

As we tried to demonstrate in the previous chapters, the design of digital work systems requires stakeholder involvement in generating relevant work knowledge, starting with articulation and proceeding with sharing and aligning it in more or less structured design spaces. When looking close to transforming organizations towards digital process support, however, the ultimate goal is to develop executable processes in evolving cyber-physical environments. Does such a scenario finally mean to educate stakeholder to become skilled in programming when designing digital work places and business processes?

Although actor-centered concepts to that direction exist, such as app'ificiation (Stary 2017), for complex domains, such as additive manufacturing, more in-depth knowledge of coding is likely to be required. To the latter direction, recent work with respect to software-intensive systems of layered approach involving various levels of abstraction has been proposed (Börger 2018). It should lead from requirements engineering to coding through abstract modeling concepts available as high-level programming constructs. Such kind of specifications help to define the code in a way stakeholders intend to, and as required to execute the corresponding software system by digital systems.

© The Author(s) 2019
S. Oppl, C. Stary, *Designing Digital Work*,
https://doi.org/10.1007/978-3-030-12259-1_9

Börger argues that the remaining gap cannot be closed by mere programming methods, but needs to be addressed by an appropriate modeling framework comprising a design and analysis method, and a language. In his understanding, programming languages must be supported by modeling at higher levels of abstraction than that of the programming language, as programming means programming reliable complex systems or software-intensive systems. The latter refer to systems where "the software and the machines which execute it are only a part of the overall system, where for the code executing computer(s) the other parts appear as environment—technical equipment, physical surrounding, information systems, communication devices, external actors, humans—upon which the behavior of the software components depends and which they affect" (ibid., p. 1).

Since we consider this kind of system as backbone of digital work design, we could look in how far coding such systems is supported by levels of abstraction, including requirements through high-level design to machine-executable code. As means of describing information on several layers of abstraction, Börger considers natural language, dedicated languages, and frameworks appropriate, when capturing programming-relevant knowledge. Of particular interest, he considers approaches which "to relate in a controllably reliable way real-world items and behavior (objects, events and actions) to corresponding items in a textual or graphical description, whether directly by code or by an abstract model that is transformed in a correctness preserving manner to code" (p. 3).

According to Börger, this epistemological problem has a communication, an evidence, and an experimental validation strand. Referring to the intrinsic properties of languages when resolving this problem, stakeholders need an understandable language. More important application such as

> language must allow the stakeholders to *calibrate the degree of precision* of descriptions (read: their level of abstraction) to the given problem and its application domain. Last but not least, the language must allow the software engineers to *link descriptions at different levels of abstraction*—transform models, lifting what compilers do to the given levels of abstraction—in a controllably correct and well documented way to code, using a practical refinement method that is supported by techniques for both, experimental validation and mathematical verification (whether informal, rigorous or formal and machine supported). (p. 2)

Börger promotes the term ground models referring to some ‘blueprints’ through which domain experts and software developers need to achieve a common understanding of a proper digital support system. This consensus serves as an essential input for validation. The intended behavior is expected to be delivered by domain experts with rigor valid in the application domain. This rigor includes describing stable domain assumption with respect to the structure and behavior of system components. That information is transformed or refined to a software system specification. It contains a sufficiently precise behavior description of the digital support system meeting the requirements as provided by the actors.

Code development requires a ground model to ensure complete and correct code (cf. Fig. 9.1). Completeness means containing all features of a system that is relevant from a behavior perspective. Correctness means conveying the meaning in a reliable way. The ground model could change in the course of code development or system evolution, leading to further development iterations. Hence, validation based on the actors’ inputs is essential.

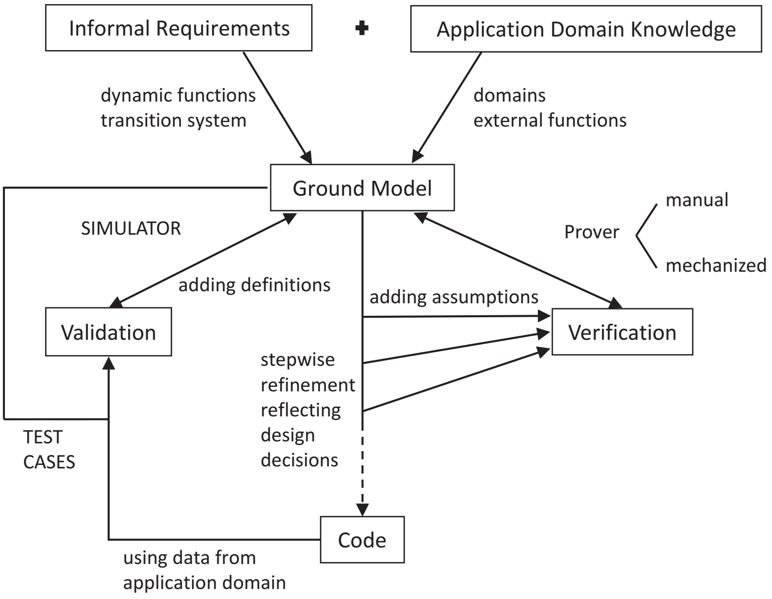

Fig. 9.1 System development involving the ground model supported by ASM (Börger and Stärk 2012)

The ground model should objectively be checkable by the respective stakeholders, in order to support articulation and alignment activities focusing on actor perspectives on work processes. Its description requires a (ground model) language which is

- generally understood
- appropriately extendable by specific application domain concepts, where needed, and
- clearly defined

It represents "a language of the kind used in rigorous scientific and engineering disciplines, made up from precise and simple but general enough basic constructs to unambiguously and directly represent arbitrary real-world facts (states of affairs) and state changing events" (Börger 2018, p. 6).

We consider the proposed organizational development framework detailed in Chap. 6 applicable to bridge the gap between articulation of stakeholder requirements and generating code. The articulation, as shown in Chap. 2, can be based on variety of formats:

- natural language which can be refined to abstract models
- graspable model entities that need to be tagged in natural language to develop a domain-relevant representation
- predefined elementary symbols representing work activities that allow complex behavior specifications based on natural language descriptions

Depending on the stakeholder capabilities and preferences, various entry points for structuring the elicitation procedure and representation of work knowledge can be selected and applied. Each of the presented formats allows further processing on a social and technical layer (Chaps. 3 and 4). For alignment and consolidation, models needs to be intelligible for all stakeholders involved in the process. They might find consensus on an abstract level or require virtual enactment to probe their model(s) of work. The latter requires executable models.

The presented concepts and instruments enable executable models to remain on an abstract, since implementation-independent level. Iterative prototyping supports is thus possible and allows for interactive validation of specific process scenarios or situations (as also advised by

(Börger 2018) for ground model inspection). Remaining on this level of description allows tracing the diagrammatic model while running its code. Behavior-centered approaches, such as Subject-oriented Business Process Management (Fleischmann et al. 2012), finally facilitate the specification of programming code, as the entire control flow and functional structure can be modeled (see Chap. 5). Still, in those cases, programming of system functions remains to be completed, either developing them from scratch or activating existing software systems. In both cases, the complexity has been reduced to a manageable system architecture.

References

Börger, Egon. 2018. Why Programming Must Be Supported by Modeling and How. In *Proceedings of ISoLA*, LNCS, vol. 11244, 1–22.
Börger, Egon, and Robert Stärk. 2012. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Science & Business Media.
Fleischmann, Albert, Werner Schmidt, C. Stary, Stefan Obermeier, and Egon Börger. 2012. *Subject-Oriented Business Process Management*. New York: Springer.
Stary, Christian. 2017. Contextual App'ification. In *Proceedings of ROCHI*.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.