Intentional Software Systems:
Engineering Intentionality in Processes of Software Development and Runtime Execution

Position Paper: 5th International Workshop on Economics-Driven Software Engineering Research

Kevin Sullivan

University of Virginia Department of Computer Science
Charlottesville, VA 22904 USA
sullivan@cs.virginia.edu

February 15, 2003

To satisfice their dynamic norms—that is, to be good enough with respect to prevailing *process*-oriented criteria such as survival, ethics, maximizing utility, or creating economic value—intentional systems must be organized to perceive, represent, processes, and respond appropriately, in the environment, to what really *matters*. In this paper, I take the position that software processes—both software design processes and computerized runtime processes—all too frequently behave badly today because they are not conceived and implemented as *intentional systems* governed by appropriately selected *dynamic norms*. The lack of attention to the issue of intentionality, in general, and to the choice of norms, in particular, often leaves such processes subject to wrong or inadequate norms, thus "unaware" of and unresponsive to what matters.

To make the notion of dynamic norms clear, consider the example of a software start-up. The developers want to restructure its code, which, having evolved somewhat messily from a prototype, is in pretty bad shape. Yet, the crucial norm for the organization is business *survival*. If satisficing this norm demands ongoing delivery of functional increments on a tight schedule, code restructuring might have to wait, no matter how attractive it is by other norms, such as the static (entity-oriented) norm, on source code structure, of "being easy to change."

Here is another example. Take the (forthcoming) IEEE Software, March 2003 feature article [1], in which, Boehm and Huang show that the widely used Earned Value Management System for controlling development processes actually has nothing do with *creating stakeholder value*. Rather, the norm governing this software development process is that *spending occur as planned*. A process can be successful by this norm and still be an utter failure by what matters. A norm that arguably would steer the process to perform better with respect to stakeholder outcomes (which, here, is what is deemed to matter) would be "continuous satisficing of all stakeholder value propositions."

More importantly, the prevailing norm fails to cause several crucial things to happen. First and foremost, the norm doesn't demand that the process recognize, monitor, or represent stakeholder value propositions. The wrong norm thus leaves the process ontologically hobbled. Second, as a result of being "unaware" of stakeholder value propositions and of the goal of satisficing them, the process is not driven to be responsive to what matters.

My position is that we need to create systems *engineered for intentionality*.[1] The first step is the choice of governing norms. The second is to confront the technical challenges of crafting perceptual, process-internal representation and processing, and response mechanisms necessary for norm-satisficing, intentional behavior. Mechanisms will generally be needed to perceive and represent material states of both process and environment, and to plan and execute normatively appropriate, adaptive behaviors. The resulting processes will seem aware of their environments, to have goals (desires), and will behave accordingly, acting and adapting appropriately over time.

Consider two examples. First, a development process under a stakeholder-satisfaction norm might represent stakeholder value propositions as evolving business cases, monitor competitors' actions, use a software

---

[1] Intentionality, in this sense, has a clearly established meaning in the philosophy of mind. The two big problems in that field are consciousness—awareness of self and other—and intentionality—to be *about* something, to have beliefs and desires.

architecture supporting schedule control as an independent variable, and drop features as needed to beat competitors to market. Second, a critical societal information system might "perceive" when it is "under attack" and reconfigure automatically to provide continuity of services *deemed* essential.

I contend that the intentional perspective holds promise for the design of software *processes* of both program development and execution. Of some interest is the observation that the approach says little about software artifacts, the traditional focus of most software engineering attention and technology. The concern is with process, behavior, and what is ultimately sought.

This idea appears to have some merit in at least three dimensions. First, it rests on seminal contemporary work in the philosophy of mind and computation—and intentionality, in particular—namely the work of Brian Cantwell Smith [3]. Second, it has potential not only to rationalize, but to bring into computer science proper—as an intentional science—a consideration of *value and values*. Third, the mechanisms needed for intentional behavior in software development and runtime processes will push the limits of existing technology in the areas of system monitoring of self and environment, internal representation including reference to relevant externals, and appropriate inference and planning techniques, based in part on mappings relating available actions to norm-denominated expected outcomes. The notion of autonomic computing, self-healing systems, and the like, clearly fall under the scope of the notion proposed here.

There is an important homonymic line of work with which the current proposal should not be confused: namely, intentional software [2], and the closely related areas of intentional programming and aspect-oriented software development. These important but quite different ideas all revolve around a specific, *static*, *technical norm*, pertaining to the relationship between two software representations: design structure and source code. The norm is that the structure of the *source code* should reflect the *design intent* of the developer. "Using Intentional Software, the actual software *source code looks like the design* [1]." The norm is static in that it applies to artifacts, not processes. It is logico-technical, rather than, say ethical or financial, insofar as it concerns structural relationships in code, not *values* (although the motivation is non-technical).

The idea presented here is fundamentally different. First, it centers on dynamic norms: that is, norms that apply to processes rather than to static artifacts. Second, it is a broader idea. To our view, it is the first formulation promising to bring considerations as diverse as ethics (such as the Rawlsian ethics of fairness underlying Boehm's Win-Win model), business value, or *values* (such as having fun), into software engineering in a general and scientific way—rather than as interesting, important, and useful, but ultimately awkward, glue-ons. Third, our idea leads to an entirely different set of issues than those addressed by intentional software. Rather than focusing on modularity, expressiveness, and structural continuity across representations at various levels of abstraction, attention turns to ontology, perception, representation, reasoning, action, and, ultimately, mattering: What material "things" does a process recognize, represent, reference, compute about, and, affect? What does it *care* about? What *matters* to it?

Finally, by making the choice of norms for synthetic processes explicit, the notion presented here provides a link between process design and human values. The question is, under what other norms do we choose the norms to impose on our designed processes? Why make Rawls's ethic of fairness the governing "law" of a development approach such as Win-Win? How about long-term shareholder value maximization? Preservation of freedom? The choice of dynamic norms fundamentally determines many key parameters of a process. Today, these considerations are most important in relation to human processes of software development. However, as our capabilities develop to create more "aware" and "autonomic" software-based systems, we might have to begin to ask about their "intentions," as well—that is, to make sure that we pick "good" ones for them. Ultimately, it's a question of what matters *to us.*

Bibliography

[1] Boehm, B. and L. Huang, "Value-Based Software Engineering: A Case Study," forthcoming, *IEEE Software*, March, 1993.

[2] Intentional Software (corporation). Front page of web site as of this writing. http://intentsoft.com/

[3] Smith, B.C., “God, Approximately,” presented at the MIT AI Lab, November 18, 1998. http://www.ageofsig.org/people/bcsmith/print/smith-godapprox4.pdf