Point – Counterpoint

Challenging Universal Truths of Requirements Engineering

Because requirements engineering is likely to be a major issue in this decade, now is a good time to examine two widely held beliefs: Requirements describe a system's "what," not its "how." Requirements must be represented as abstractions.

JAWED SIDDIQI, *Sheffield-Hallam University*

To trace the emergence of significant ideas in software development, simply track what part of the idealized life cycle has gotten all the attention over the years: In the '60s, it was coding; in the '70s, design; in the '80s, specification.

Of course, all these issues have always been important and major contributions are not restricted to these decades, but it was during these times that each activity emerged as a subject in its own right, with its own community of activists.

I think most would agree that requirements engineering will be a critical issue in the '90s. And it has certainly attracted its share of missionaries! In this essay, I explore two of the so-called universal truths of requirements engineering, to stimulate discussion and challenge current thinking.

FIRST UNIVERSAL TRUTH

Requirements describe the "what" of a system, not the "how."

It is generally accepted that a requirements document contains a description of what the system will do without describing how it will do it.[1] Alan Davis calls this the "what-versus-how paradox:" Most popular approaches to requirements engineering solve the problem of requirements engineering in a unique way.

We can make sense of this paradox by observing that each solution is one level in a hierarchy of abstraction levels, distinguished by the extent that it abstracts requirements. As Davis said, "one person's 'how' is another person's 'what.'" An intermediate level is both the "how" of the level above it and the "what" of the level below it.[1]

This appears to solve the problem, but it leaves unresolved the question of whether or not it is possible or desirable to separate the "what" from the "how" in practice.

The essence of this evokes the debate over structured programming in the 1970s. Then, structured-programming proponents like Edgser Dijkstra argued with firm conviction that developing, understanding, and reasoning about programs on the basis of assertions (the "what") was superior to using procedural thinking (the "how"). This despite empirical evidence that not only did people understand programs from an operational — or at least from a mixed — perspective, but also that programming was a process involving experimentation and it could not be reduced to a rigid ideology based on abstractions.

This debate spilled over into the 1980s, when the issue was whether or not specifications should be executable. In this case, Ian Hayes and Cliff Jones strongly recommended keeping the "what" and "how" separate.[2]

The declarative approach to requirements is overwhelmingly more popular than what Harold Abelson and Gerald Sussman have called the imperative approach, which they distinguish from the

0740-7459/94/$04.00 © IEEE

"classical mathematical" viewpoint embodied in declarations.[3]

Others have questioned the entire premise of the what-how debate. William Swartout and Robert Balzer have written that separating the specification from the implementation is "overly naive and does not match reality.... Specifications and implementations are, in fact, intimately intertwined because they are respectively, the already-fixed and yet-to-be done portions of multistep development."[4]

In a similar vein, Joseph Goguen has argued "the belief that the steps of a life cycle should be executed sequentially is a crude form of the myth that there is more or less a unique best system to be built."[5] According to Goguen, it is better to think of requirements as "...emergent, in the sense that they do not already exist, but rather emerge from interactions between the analyst and the client organization."

SECOND UNIVERSAL TRUTH

Requirements should be represented as abstractions.

Requirements modeling involves using abstractions to produce a view of the system that is independent of the method and notation used. Indeed, implicit in Davis' exposition of the what-how paradox is the notion that all models vary only in their level of decomposition. This implies that there is some objective reality that can be abstracted. It also implies that requirements methods are free of assumptions.

Matthew Bickerton and I have contradicted these implied characteristics.[6] We believe that assumptions about things like organizations and society invariably become embedded in the requirements method as it is developed. Therefore, not only are such methods not assumption-free, their application cannot result in the same solution.

WE SHOULD FOCUS ON INTERACTIONS AMONG USERS AND NOT ON INDIVIDUAL USERS.

However, when we place today's most popular methods into a taxonomy, they all tend to fall into the same class: rational functionalism.[6] Ironically, they at least *appear* to be based on similar assumptions.

But what of the notion of an objective reality that can be captured in some abstraction? This begs the question: Whose reality? Some argue that reality is socially constructed as a result of interactions among participants in the requirements process.[7] Therefore, the *constructed reality* varies from participant to participant — no single group or even participant knows or owns the abstracted model.

So if we reject the traditional concept of an abstract model — a tool that is integral to most scientific disciplines — what are we left with?

We are left with the view that requirements elicitation should not be based on capturing the needs of individual users. Instead, it should focus on the interaction of participants (social) rather than individual participants (cognitive).

Along these lines, Goguen and Charlotte Linde have enumerated the limitations of traditional elicitation techniques (interviews, questionnaires, and protocol analyses) and propose that we can improve accuracy by using conversational, interaction, and discourse analyses instead.[8]

Does the requirements-engineering community need to completely reorient itself toward this new, social, integrated perspective? No, but I am suggesting that adopting a social perspective will let us uncover elements that a purely technical perspective will miss.

There is this conundrum: The social perspective requires that we ground our observations in real-world settings, yet system development requires formalism and abstraction. *This* then is the thorny problem facing requirements engineering. ◆

ACKNOWLEDGMENTS

Many of these ideas have originated from my close collaboration and friendship with Alan Davis and Joseph Goguen — I am indebted to both. Any misrepresentation of their ideas is, of course, completely my responsibility.

REFERENCES

1. A.M. Davis, *Software Requirements: Objects, Functions and States*, Prentice-Hall, Englewood Cliffs, N.J., 1993.
2. I.J. Hayes and C.B. Jones, "Specifications are Not (Necessarily) Executable," *Software Eng. J.*, Nov. 1989, pp. 330-338.
3. H. Abelson and G. Sussman, *The Structure and Interpretation of Computer Programs*, McGraw-Hill, New York, 1985.
4. W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specifications and Implementations," *Comm. ACM*, July 1982, pp. 438-440.
5. J. Goguen, "Requirements Engineering: Reconciliation of Technical and Social Issues," tech. report, Centre for Requirements and Foundations, Oxford University Computing Lab, Cambridge, U.K., 1992.
6. M. Bickerton and J. Siddiqi, "The Classification of Requirements Engineering Methods," *Proc. Int'l Symp. Requirements Eng.*, IEEE CS Press, Los Alamitos, Calif., 1993, pp. 182-186.
7. C. Floyd et al., *Software Development and Reality Construction*, Springer Verlag, Berlin, 1991.
8. J.A. Goguen and C. Linde, "Techniques for Requirements Elimination," Proc. Int'l Symp. Requirements Eng., IEEE CS Press, Los Alamitos, Calif., 1993, pp. 152-164.

Jawed Siddiqi is the director of the Computing Research Centre at Sheffield-Hallam University. He is also a visiting researcher at the Centre for Requirements and Foundations at the Oxford University Computing Laboratory. His research interests are software engineering, the human factors of software development, requirements engineering, and animation of formal specification.

Siddiqi received a BS in mathematics from the University of London and an MS and a PhD in computer science from the University of Aston in Birmingham. He is a member of the British Computer Society and the IEEE Computer Society.

His address is Sheffield-Hallam University, School of Computing and Management Science, Hallamshire Business Park, 100 Napier St., Sheffield S11 8HD, UK; j.l.siddiqi@shu.ac.uk.