Editor: Robert L. Glass ■ Computing Trends ■ rlglass@acm.org

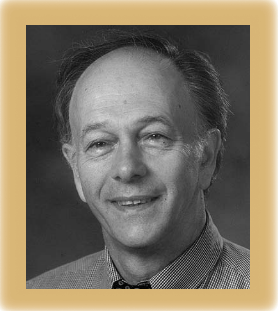
Climbing over the "No Silver Bullet" Brick Wall

R. Geoff Dromey

… in which I oppose the notion that we can't hope to make significant gains in software development.

A good representation is usually the last thing you think of, not the first!" (with apologies to Harlan Mills, who spoke of "design" rather than "representation.")

The way we represent things often has significant consequences. We commonly find that the choice of representation seriously impacts the complexity and relative difficulty of a task, the ease of understanding and changing what is represented, and the likelihood of making mistakes.



Do software properties block progress?

The language representation and means of composition used in the early programming languages (and still retained in current languages) have influenced people in the software engineering community to form certain views about software that impede progress with system modeling. Frederick Brooks, in his influential "No Silver Bullet" article, comments on software's essential nature: "The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions."[1] Taking his lead from a classification that Aristotle used, Brooks assesses the prospects of substantially improving software technology by dividing its difficulties into two categories:

- properties inherent to software and
- accidents or artifacts of the current state of the technology's evolution.

He sees complexity, conformity, changeability, and invisibility as the inherent properties.

Brooks goes on to comment that "surely the most powerful stroke for software productivity, reliability, and simplicity has been the progressive use of high-level languages for programming." He then asks, "What does a high-level language accomplish?" His answer is that "it frees a program from much of its accidental complexity." If he's right, and we've gotten rid of most of the accidental complexity that software imposes, then the best we can hope for is only slow, incremental improvement in our software engineering capability.

According to Brooks, another serious impediment to advancing the discipline is that "software is invisible and unvisualizable." To use his terminology, software doesn't let us capture a geometric reality in a geometric abstraction, as we do with other physical systems. He concludes, "The reality of software is not inherently embedded in space." Tony Hoare made the related observation that almost all complex man-made structures (software aside) possess the properties of clear spatial separation and spatial organization of their components.[2]

Continued on p. 118

Lawrence H. Putnam, Quantitative Software Management
Hridesh Rajan, Iowa State Univ.
Guus Ramackers, Oracle
Subburaj Ramasamy, Electronics Test & Development Centre
Jeremy Rand, Alcatel
Anand Ranganathan, Univ. of Illinois, Urbana-Champaign
Awais Rashid, Lancaster Univ.
Bjorn Regnell, Lund Univ.
Donald Reifer, Reifer Consultants
Ralf Reussner, Univ. of Oldenburg/OFFIS
Bill Riddle, Software Deployment Affiliates
Stan Rifkin, Master Systems
Suzanne Robertson, Atlantic Systems Guild
Martin Robillard, McGill Univ.
Rob Rodgers, Northrop Grumman IT
Eelco Rommes, Philips Research
David Rosenblum, Univ. College, London
Mark Roth, Science Applications Int'l
Gregg Rothermel, Oregon State Univ.
Terence Rout, Griffith Univ.
Walker Royce, IBM
Ioana Rus, Univ. of Maryland
Hossein Saiedian, Univ. of Kansas
Julio Cesar Sampaio do Prado Leite, Pontifícia Univ. Católica do Rio de Janeiro
Arno Schmidmeier, AspectSoft
Robert Schwanke, Siemens
Sahra Sedighsarvestani, Univ. of Missouri-Rolla
Ed Seidewitz, Data Access Technologies
Bran Selic, IBM Software Group
Bikram Sengupta, IBM India Research Lab
Johanneke Siljee, Univ. of Groningen
Alberto Sillitti, Free Univ. of Bozen
Nivedita Singhvi, IBM
Dennis Smith, Carnegie Mellon Univ.
Harold Smith III, Penn State Univ. New Kensington
Angela Sodan, Univ. of Windsor
Martin Solari, Universidad ORT Uruguay
Rini Solingen, LogicaCMG
Diomidis Spinellis, Athens Univ. of Economics and Business
Judith Stafford, Tufts Univ.
Michael Stal, Siemens
Bernhard Steffen, Universität Dortmund
Dominik Stein, Univ. of Duisberg-Essen
Magdin Stoica, EngPath
Wolfgang Strigel, QA Labs
Christoph Strnadl, Atos Origin IT
Paul Strooper, Univ. of Queensland
Eleni Stroulia, Univ. of Alberta
Giancarlo Succi, Free Univ. of Bolzano-Bozen
Mario Sudholt, EMN/INRIA
Kevin Sullivan, Univ. of Virginia
Håkan Sundell, Chalmers Univ. of Tech
Alistair Sutcliffe, Centre for HCI Design
Stanley Sutton, IBM T.J. Watson Research Center
Clemens Szyperski, Microsoft
Mini TT, Philips Software Centre
Nejmeddine Tagoug, United Arab Emirates Univ.
Mehdi Baradaran Tahoori, Northeastern Univ.
Bedir Tekinerdogan, Bilkent Univ.
Thomas Thelin, Lund Univ.
Steffen Thiel, Robert Bosch Corporation
Martyn Thomas, Martyn Thomas Associates
Stuart Thomason, Keele Univ.
Ciprian Ticea, QA Labs
Scott Tilley, Florida Inst. of Technology
Steve Tockey, Construx Software
Paolo Tonella, Istituto per la Ricerca Scientifica e Tecnologica
Marco Torchiano, Politecnico di Torino
Kal Toth, Portland State Univ.
Will Tracz, Lockheed Martin
Laurence Tratt, King's College London
Richard Turner, Systems and Software Consortium
Virpi Tuunainen, Helsinki School of Economics
Jeffrey Tyree, Capital One Financial
Naoyasu Ubayashi, Kyushu Inst. of Technology
Sebastian Uchitel, Imperial College London
Ricardo Valerdi, Univ. of Southern California
Klaas van den Berg, Univ. of Twente
Frank van der Linden, Philips Medical
Wim Vanderperren, Vrije Universiteit Brussel
William van der Sterren, Philips Medical
Jan van der Ven, Univ. of Groningen
Arie van Deursen, CWI and Delft Univ. of Tech
Pascal Van Eck, Univ. of Twente
Rob Van Ommering, Philips Research
Hans van Vliet, Free Univ.
Tathagat Varma, Network Associates India
Alexandre Vasseur, BEA Systems
Sira Vegas, Universidad Politecnica de Madrid
Belen Vela Sanchez, Univ. Rey Juan Carlos
Jeffrey Voas, SAIC
Markus Voelter
Robert Walker, Univ. of Calgary
Dean Wampler, Rational Software
Julie Waterhouse, IBM
Matthew Webster, IBM UK
Elaine Weyuker, AT&T Laboratories Research
David Whitlock, Portland State Univ.
James Whittaker, Florida Tech
David Wile, Teknowledge
Eric Wohlstadter, Univ. of British Columbia
Alexander L. Wolf, Univ. of Colorado
Eric Wong, Univ. of Texas at Dallas
Kenny Wong, Univ. of Alberta
Eoin Woods, UBS Investment Bank
Simon Wright, Integrated Chipware
Tao Xie, North Carolina State Univ.
Alec Yasinsac, Florida State Univ.
Michal Young, Univ. of Oregon
Trevor Young, Univ. of British Columbia
Yuen Tak Yu, City Univ. of Hong Kong
Marvin V. Zelkowitz, Univ. of Maryland
Peter Zimmerer, Siemens

Continued from p. 120

If we accept these arguments, where does this leave us? In summing up his assessment of the prospects for software engineering, Brooks suggests it's unlikely that there will be any "inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware." In other words, "building software will always be hard. There is inherently no silver bullet"—we've run into a brick wall.

Scaling the wall

Faced with a situation like this, our greatest challenge in advancing any discipline is always to break free from the shackles of our past. In this regard, David Harel's advice provides a signpost to where software engineering is and should be heading: "It is our duty to forge ahead to turn systems modeling into a predominantly visual and graphical process."[3]

What Brooks calls the "essence" of software entities has little to do with the conceptual view of systems. Systems are built out of a network of interacting components (some of which might be systems in their own right). Such a view implies all systems might have designs that can be embedded in space. It doesn't matter whether we're talking about systems we intend to implement in software, hardware systems, other physical systems, business systems, or any other conceptual systems. In all cases, the system components encapsulate and exhibit individual behavior, and they interact by passing control and data to other components. This results in the overall system exhibiting integrated behavior.

An appropriate representation of this behavior can provide the ladder that lets us climb over the brick wall—to get complexity and change under control, to overcome the so-called invisibility of software, to make gains with conformity, and, as a side benefit, to detect requirements problems early. With a suitable behavioral representation, we can systematize and simplify the task of going from a set of requirements to a design. We can consider individual functional requirements to represent fragments of behavior, while a design that satisfies a set of functional requirements represents integrated behavior. This perspective enables us to construct a design out of its requirements.

The behavior-tree ladder

A formal representation called *behavior trees* makes this possible,[4] thereby removing a lot of accidental complexity from the analysis and design phases. Behavior trees of individual functional requirements (constructed by rigorous, intention-preserving translation from their natural-language representation) can be composed, one at a time, to create an integrated design behavior tree that serves as a system's formal behavior specification. Because we only have to deal with one requirement at a time, the task's complexity greatly decreases. From this problem domain representation, we can then transition directly, systematically, and repeatably to a solution domain representation of the system's architecture (its component integration specification) and the behavior designs of the system's individual components—both are emergent properties of the integrated design behavior tree.[4] We can then implement the component behavior designs (using design-by-contract) and directly convert the diagrammatic form of the architecture to an implementation, using a one-to-one mapping. The result is an implementation in which the components and interactions dominate. This is the best we can do to embed the architecture in the implementation—the goal we always seek when designing and implementing physical systems. Any other architecture implementation strategy is likely to introduce unnecessary accidental complexity.

If we take the line of attack I've outlined, what progress do we make against Brooks' inherent properties of software—complexity, changeability, invisibility, and conformity—and the vexing problem of requirements defects? We make significant progress with complexity because we only need to focus on the detail in one requirement at a time, greatly reducing the load on our short-term memory. Change also becomes easier. If a system needs a new requirement, we simply translate that requirement to a behavior tree, integrate it into the design behavior tree, and carry out the systematic steps to obtain the modified integration specification and component behaviors.[5] Invisibility is alleviated because we can embed the architecture in the implementation. Conformity is addressed by using a single behavioral representation for requirements, the design, and the individual component designs and by completely separating the integration of components, as defined by the design behavior tree, from the implementation of components. And finally, we find requirements defects when we translate requirements to behavior trees and integrate the behavior trees. We can also perform a number of systematic checks, including model-checking on the integrated design behavior tree to find still other defects. When we complement these strategies with an integrated view of a system's compositional and data requirements, we have all the information we need to fully support the design (not discussed here).

High-level programming languages might have helped remove accidental complexity at one level. However, much accidental complexity remains in most analysis and design processes and in requirement and design representations. Simpler, more well-defined processes and better, simpler representations hold the key to further substantial advances in software engineering. This I've tried to do using behavior trees.

References

1. F.P. Brooks Jr., "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, 1987, pp. 10–19.
2. C.A.R. Hoare, "Programming: Sorcery or Science?" *IEEE Software*, vol. 1, no. 2, 1984, pp. 5–16.
3. D. Harel, "Biting the Silver Bullet," *Computer*, vol. 25, no. 1, 1992, pp. 8–20.
4. R.G. Dromey, "From Requirements to Design: Formalizing the Key Steps," *Proc. 1st Int'l Conf. Software Eng. and Formal Methods* (SEFM 03), IEEE CS Press, 2003, pp. 2–11.
5. L. Wen and R.G. Dromey, "From Requirements Change to Design Change: A Formal Path," *Proc. 2nd Int'l Conf. Software Eng. and Formal Methods* (SEFM 04), IEEE CS Press, 2004, pp. 104–113.

R. Geoff Dromey is a professor at Griffith University and the director of the university's Software Quality Institute. Contact him at g.dromey@griffith.edu.au.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1314; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Subscription rates: IEEE Computer Society members get the lowest rate of US$46 per year, which includes printed issues plus online access to all issues published since 1988. Go to www.computer.org/subscribe to order and for more information on other subscription prices. Back issues: $20 for members, $128 for nonmembers (plus shipping and handling). This magazine is available on microfiche.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855-1331. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.