

Self-Renewing Applications



Munindar P. Singh • North Carolina State University

In computer science, the word *application* carries two meanings I'd like to explore here. The first is the traditional meaning in which an application is any program that runs on top of an operating system. This program could be something like Microsoft Word or Mozilla Thunderbird.

A program is, of course, not an application. This is where the second and – dare I say – more sensible meaning comes into play. An application is a usage. A ceiling fan is an application of electrical motor technology. Person-to-person communication is an application of radio technology.

Software engineering has become good at building applications-as-programs but not applications-as-uses. Of course, software engineering is about building and maintaining programs, and I'm not arguing to change this. Moreover, any software engineer worth his or her salt will agree – and possibly insist on – eliciting clean requirements describing how and where their program will be used. But in software engineering, as in the rest of computer science, the application-as-use idea plays second fiddle to the application-as-program one.

Software practice has suffered from and continues to suffer from many shortcomings as a result. Programs are difficult to design and build; they often fail to satisfy user requirements. If they work adequately at all, it's more often due to users adapting to the program than the program meeting users' requirements.

The thesis of this column is that software engineering would be well served if we began to think of application-as-use as primary. In particular, if we could develop user interactions correctly, the application-as-use of a software artifact would help renew that artifact.

Software Engineering

Let's consider leading software development approaches. The classical waterfall approach

proceeds as follows. Capture all requirements, specify, design, implement, test, and last cross your fingers and hope it flies. Only rarely would developers apply the waterfall approach quite as rigidly as this characterization suggests, but it is true in spirit.

What we see more often is the traditional spiral approach, which proceeds somewhat like the following. Capture some requirements, specify, design, implement, test, validate and expand requirements, and iterate. The newer agile approaches make the spiral tighter, but they don't fundamentally alter it.

Software development approaches are valuable but rather micro in their orientation. At first blush, software engineering as a micro activity might seem like a contradiction in terms. But software engineering is only a small part of the overall value network. Remember, there are users, too. For most successful software products, the number of users far exceeds the number of developers – about the only exceptions might be one-of-a-kind engineering efforts such as NASA's. This is understandable – in most settings, a gain in productivity is software's primary value proposition.

Renewal via Application

I claim that any artifact put to good use must represent the user's needs and changing situation in some form. Let's call such a representation a *model*. A model might be explicit, as in a database schema, or implicit, as in a hard-coded prejudice about how the user needs a screen laid out. We can trace most software artifact shortcomings to poor (that is, erroneous or incomplete) models. The trouble with models is that they tend to drift from reality. Users' needs change – and most often you don't know those needs well enough to begin with.

The common feature of existing software engineering approaches is that the construction of a software program – or generally any artifact – precedes its use. Whether the construction happens all in one shot or incrementally is irrelevant for our present purposes. Thus the approaches I've mentioned tacitly assume that their models will either be perfect at the outset (waterfall) or evolve through repeated developer intervention.

Recalling that there are – or ought to be – more users than developers for almost any software product, and further recognizing that the users' needs continually evolve, it stands to reason that the construction and maintenance of these models by developers simply can't keep up.

This leads us to the vision of what I call self-renewal. To accomplish self-renewal, we must design our software artifacts so that the act of using them updates and expands their models. This presupposes that the models are clear and explicit and that the application-as-program supports user manipulation. Self-renewal as I propose it is inextricably tied to Internet computing because it relies fundamentally on interactions between users and models, and especially between multiple users. Let me be the first to admit, however, that this vision is still quite incomplete and that we have a long way to go before self-renewal in its general sense can become practical.

Simple forms of self-renewal have been emerging for a few years.

Today's Web 2.0 approaches illustrate limited levels of self-renewal. These approaches help users collaborate to maintain a rather minimal common model. An example is category labels or tags on pictures, for instance, as supported and promoted by websites such as Flickr (www.flickr.com). In *social tagging*, users assign tags to pictures to facilitate subsequent search by others. Users can see the tags others have applied on the pictures they like, and can thus choose the popular tags for their pictures. Because users have an interest in having others find their pictures, they have an incentive to choose tags that are popular. In this manner, popular tags gain additional use. Thus, with respect to the tags, Flickr supports the self-renewing application of picture sharing by

IEEE Internet Computing

Editor in Chief

Michael Rabinovich • misha@eecs.cwru.edu

Associate Editors in Chief

M. Brian Blake • mb7@cse.nd.edu
Siobhán Clarke • siobhan.clarke@cs.tcd.ie
Maarten van Steen • steen@cs.vu.nl

Editorial Board

Virgilio Almeida • virgilio@dcc.ufmg.br
Elisa Bertino • bertino@cerias.purdue.edu
Azer Bestavros • best@cs.bu.edu
Vinton G. Cerf • vint@google.com
Fred Douglass* • f.douglass@computer.org
Schahram Dustdar • dustdar@infosys.tuwien.ac.at
Stephen Farrell • stephen.farrell@cs.tcd.ie
Robert E. Filman* • filman@computer.org
Juliana Freire • juliana@cs.utah.edu
Carole Goble • cag@cs.man.ac.uk
Michael N. Huhns • huhns@sc.edu
Barry Leiba • barryleiba@computer.org
Samuel Madden • madden@csail.mit.edu
Cecilia Mascolo • cecilia.mascolo@cl.cam.ac.uk
Pankaj Mehra • pankaj.mehra@ieee.org
Chris Metz • chmetz@cisco.com
Dejan Milojičić • dejan@hpl.hp.com
George Pallis • gpallis@cs.ucy.ac.cy
Charles J. Petrie* • petrie@stanford.edu
Gustavo Rossi • gustavo@lifia.info.unlp.edu.ar
Amit Sheth • amit.sheth@wright.edu
Munindar P. Singh* • singh@ncsu.edu
Oliver Spatscheck • oliver@spatscheck.com
Torsten Suel • suel@poly.edu
Craig W. Thompson • cwt@uark.edu

Shengru Tu • shengru@cs.uno.edu
Doug Tygar • tygar@cs.berkeley.edu
Steve Vinoski • vinoski@ieee.org
* EIC emeritus

CS Magazine Operations Committee

Dorée Duncan Seligmann (chair), Erik Altman, Isabel Beichl, Krish Chakrabarty, Nigel Davies, Simon Liu, Dejan Milojičić, Michael Rabinovich, Forrest Shull, John R. Smith, Gabriel Taubin, Ron Vetter, John Viega, Fei-Yue Wang, Jeffrey R. Yost

CS Publications Board

David A. Grier (chair), Alain April, David Bader, Angela R. Burgess, Jim Cortada, Hakan Erdogmus, Frank E. Ferrante, Jean-Luc Gaudiot, Paolo Montuschi, Dorée Duncan Seligmann, Linda J. Shafer, Steve Tanimoto, George Thiruvathukal

Staff

Editorial Management: Rebecca Deuel-Gallegos
Lead Editor: Linda World, lworld@computer.org
Editorial Business Operations Manager: Robin Baldwin, rbaldwin@computer.org
Publications Coordinator: internet@computer.org
Contributors: Thomas Centrella, Molly Gamborg, Greg Goth, and Nancy Talbert

Director, Products & Services: Evan Butterfield
Senior Manager, Editorial Services: Lars Jentsch
Manager, New Media & Production: Steve Woods
Senior Business Development Manager: Sandy Brown
Membership Development Manager: Cecelia Huffman
Senior Advertising Supervisor: Marian Anderson, manderson@computer.org

Technical cosponsor:



IEEE Internet Computing
IEEE Computer Society Publications Office
10662 Los Vaqueros Circle
Los Alamitos, CA 90720 USA

Editorial. Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Internet Computing* does not necessarily constitute endorsement by IEEE or the IEEE Computer Society. All submissions are subject to editing for style, clarity, and length.

Submissions. For detailed instructions, see the author guidelines (www.computer.org/internet/author.htm) or log onto *IEEE Internet Computing's* author center at ScholarOne (<https://mc.manuscriptcentral.com/cs-ieee>). Articles are peer reviewed for technical merit.

Letters to the Editors. Email lead editor Linda World, lworld@computer.org

On the Web. www.computer.org/internet/

Subscribe. Visit www.computer.org/subscribe/.

Subscription Change of Address. Send requests to address.change@ieee.org.

Missing or Damaged Copies. Contact help@computer.org.

To Order Article Reprints. Email internet@computer.org or fax +1 714 821 4010.

IEEE prohibits discrimination, harassment, and bullying. For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

letting users maintain a model – a set of tags constituting a rudimentary vocabulary.

A related but distinct idea is *crowdsourcing*. In its current forms, crowdsourcing involves a *task manager* (a user or a company) who first persuades (through suitable incentives) several people to work on a task and then combines their results to produce an overall solution. That is, the task manager drives the entire process, including aggregating results. A recent crowdsourcing success was the DARPA balloon-locating challenge that a team from MIT won (<https://networkchallenge.darpa.mil/Default.aspx>).

These two approaches offer some interesting contrasts. Flickr arguably illustrates a better architecture. It's bottom up, and although it's clear that the Flickr site has something to do with the task, users themselves initiate picture uploading and assign tags. Crowdsourcing today is top-down but offers the possibility of supporting far greater structure in the results than does tagging. By and large, both these approaches help find consensus among users.

I envision self-renewing applications as a generalization of these approaches: the renewal might not be driven by one party, the models they build would generally carry greater structure, and the participants might well be strategic – that is, have an interest in the outcome beyond the explicit incentives offered to them to provide the true (honest and accurate) answer. I expect that an essential basis of self-renewal will reside in the interactive nature of the desired uses and how they influence the underlying model. If a user is merely supposed to construct a model that a software developer would otherwise construct, no gain in productivity would occur: a trained software developer could do a better job and faster than a typical user.

Users would self-organize into communities of practice and continually develop and refine through applications-as-programs the models needed for their applications-as-uses. In the business process management setting, users (administrative staff) would identify best practices and thus refine their business process by exercising it. In the healthcare setting, users (physicians) can create clinical guidelines refined for patients with related ailments but differing attributes. In a marketing setting, users (analysts) might create analytical queries and marketing campaigns that refine and aggregate key market segments. What these examples have in common is the largely cooperative but also competitive creation of structured practical knowledge, exactly as has been happening in communities of practice since before the dawn of computing, but at greater scales of speed, depth, and size.

Challenges

The vision of self-renewal is a natural outgrowth of the notion of the pragmatic Web, which I've been advocating for nearly a decade.¹ But it's far from mature as an idea and raises more questions than answers. Today's approaches involve interactions that are flat and yield models that are unstructured. How might we expand the variety of interactions to incorporate richer and truer organizational models? How can we develop and maintain models that involve greater subtlety than simple aggregations and statistics, as in the current approaches? How might we support multiple user perspectives along with consensus – in particular, how might we deal with self-interested users? These aren't clear-cut technical problems at this stage but can form the basis of a new research agenda that gives more than lip service to the importance of users and uses.

Here's a quick way to contrast self-renewal with current software engineering approaches:

- *Traditional* – build it, and they will come.
- *Agile* – build it partially, and they will come; build it some more, and some more will come.
- *Self-renewing* – they will come, and they will build it.

To accomplish self-renewal requires not just improvements in model representations and model-driven software but also a change in attitude. Today's software approaches are paternalistic, and this paternalism pervades our profession. We continually seek to impose “correct” solutions on users. If self-renewal is to take hold, paternalism must go. Users organizing into ad hoc communities of practice should be able to determine the solutions that they can best put to use. After all, this is what it means for anything to be an application. □


Acknowledgments

I'm indebted to Michael Huhns and Amit Chopra for comments on a previous version.

Reference

1. M.P. Singh, “The Pragmatic Web,” *IEEE Internet Computing*, vol. 6, no. 3, 2002, pp. 4–5.

Munindar P. Singh is a professor of computer science at North Carolina State University. His research interests include multi-agent systems applied in social networks and organizations. Singh is a fellow of IEEE and a former editor-in-chief of *IEEE Internet Computing*. Contact him at singh@ncsu.edu.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.