SCM Branching Strategies

To the Editor:

We read with great interest the article by Chuck Walrad and Darrel Strom titled "The Importance of Branching Models in SCM" (Sept. 2002, pp. 31-38). We also participated with the authors in an online discussion of the article at www.cmcrossroads.com.

The authors described a common branching mishap and used branch-by-release as a way of illustrating it. We and others who participated in the cmcrossroads.com discussion are concerned that the article gave branch-by-release a "bad rap" by implying that its use will always cause the problem the authors describe. It need not.

We agree that the solution and problems discussed in the article are both real and common, and understanding them is very important. Attributing those problems to branch-by-release "throws out the baby with the bathwater." The true culprit is the absence of mainlining—without which, branching-by-purpose suffers the same fate. Whether branching by purpose or by release, proper mainlining overcomes these problems in practice. In fact, the branch-by-purpose model that the authors propose does exactly this.

We are concerned that this misattribution does a disservice to the practicing SCM community, and it might have been averted by familiarity with literature on the specific subject of branching strategies. A Google search on "branching strategies" reveals several sources, including "Advanced SCM Branching Strategies" by Steven Vance (http://www.perforce.com/perforce/conf98/abstracts.html#branching), "High-Level Best Practices in Software Configuration Management" by Laura Wingerd

and Christopher Seiwald (http://www.perforce.com/perforce/conf98/abstracts.html#best_practices), and "Streamed Lines: Branching Patterns for Parallel Software Development" (http://acme.bradapp.net/branching). We were disappointed that a recent article on branching strategies neglected to mention any of these papers, and hoped that *Computer*'s submission review process would have flagged the absence of references to papers about the specific topic of the article.

Brad Appleton
brad@bradapp.net
Michael Sayko
mss@acm.org

The authors respond:

What we have here is a failure to communicate because of the differences in our nomenclature and the terminology that Brad Appleton uses on his Web site and in his recently published book (not available when we submitted our article last April). The designation of a continuous mainline is one of the basic premises of our branch-by-purpose model—one that we failed to enunciate in our article, for which we apologize.

Mr. Appleton's comment about "throwing the baby out with the bathwater" misses the point we tried to make in our description of the branch-by-release model. Those without experience in SCM often fall into the branch-by-release model—pure "cascading" without "mainlining." We used branch-by-release for this because it reflects the typical default case—establishing the latest release as the new code base and going forward from it. This leads to the troubles we describe. When you understand the issues and add mainlining, it is no longer the branch-by-release model we described, but something closer to our branch-by-purpose model.

We did indeed look at the literature, focusing on branching as reflected in adopted or emerging standards, which we cited. As for the Perforce Web site paper, we have found over the years that in many cases vendors use their own particular terminology and have their own biases. For that reason, among others, we tend not to reference papers written by vendors.

Mr. Appleton's book and his "Streaming Lines" discussion on his Web site deserve attention. It would be a pleasure to see him become involved in IEEE standards work.

SOFTWARE DEVELOPMENT DIVISION OF LABOR

To the Editor:

In "Jobs, Trades, Skills, and the Profession" (The Profession, Sept. 2002, pp. 104, 102-103), Neville Holmes states that blaming the IT industry or a servile government for using false claims of a software labor shortage to justify hiring foreign workers is unfair. Holmes contends that the main fault for this deceit lies with the computing profession. However, the remainder of the article does not seem to justify this statement. Perhaps Holmes intended to argue that, since the profession makes no distinction between engineers and tradespeople, neither do employers. Since all software developers are, therefore, equal, why not use the cheapest option available?

While I agree with much of what Holmes says about how labor should be divided in software development, I believe that other factors lead to the use of the cheapest labor. It has been known for decades that the small percentage of developers who would qualify as engineers are orders of magnitude more effective at developing designs than the rest of the population. I think many software development organizations know this and deliberately ignore it. It makes management easier and increases profits.

Large software development projects for the US government, for example, often make more money for the contractor the longer they take. Hiring large numbers of incompetent but cheap coders is more profitable than doing things correctly. An engineer who works on such a project and tries to do things correctly is often labeled a troublemaker, and is removed from the project as quickly as possible.

Commercial software products, such as word processors and spreadsheets, are more profitable if they are badly developed. A company that creates and sells the perfect word processor, for example, would have large sales initially, followed by declining sales, and would no doubt soon go out of business. A badly written program, on the other hand, continues to generate revenue forever as users are forced to buy upgrades to correct errors.

There appear to be significant market forces against the kind of division of labor Holmes advocates. Until those forces are removed, change is unlikely.
Jeffrey R. Carter
Apache Junction, Ariz.
jrcarter@acm.org

Neville Holmes responds:

The issue here is why those significant market forces prevail. The IT industry can hardly be expected, under present conditions, to ignore the financial advantage of those forces. Likewise, the governments of developed nations, unashamedly basing their policies on promoting their economies, can hardly be expected to oppose market forces.

So who should take responsibility for allowing the unprofessional practices and unsubstantiated claims behind the IT industry's market forces to prevail unchallenged, if not the computing profession?

To the Editor:

Neville Holmes's analysis in "Jobs, Trades, Skills, and the Profession" is perfectly true, but incomplete. While the division between engineering, construction, and the use of informatics products is not yet either globally accepted or formally defined, I think this will be the natural evolution of the field.

Compared with human evolution, automatic computing is in its infancy. At this stage, every activity is a mix of art, trade, and work.

The ancient Greeks discovered the electrical phenomenon. Later, some physicists formalized mathematically the basics of this science, providing the basis for the engineering discipline. In every human group—and also among some nonhumans—building a home is a common practice that has nothing to do with engineering. However, the engineering profession evolved from a formalization of methods for accomplishing this task.

We do not "impose" data on materials. Rather, data provides a way to define and manage materials. When data management becomes sufficiently complex, a full engineering discipline will emerge.

Programming is an art, a craft, and—for too many—a trade. But a programmer is to a data engineer like a mechanic is to an automotive engineer. The distinction between being a mechanic on the Ferrari F1 team and doing repairs in an automobile repair shop is not a difference in the nature of the work but in the career path. Engineers are skilled complexity managers in the technical field, not just persons who like to play with engines.

Over time, a true data engineering profession will emerge that has many branches, and data engineers will have formal and well-known methods to use for defining data-related projects. This is different from both programming and using programs.
Franco Favento
Trieste, Italy
f.favento@ieee.org

Neville Holmes responds:

On his point about automatic computing being in its infancy, I would refer Mr. Favento to my October 2001 column ("The Profession's Future Lies in Its Past," pp. 120, 118-119). On the history of engineering, I think it is at least ironic that engineering was for a long time only a military discipline. The first branching out was therefore called civil engineering, from which other branches of engineering evolved within the past 100 years.

On the point of imposing data on materials, this springs from the official—and more or less ignored—definition of data that I discussed in my May 2001 column ("The Great Term Robbery," pp. 96, 94-95). Simone Santini also addressed this issue in The Profession column in December 2002 ("Using Language More Responsibly," pp. 128, 126-127).

On the question of data engineering, a data engineer typically works with other professionals because they are the main users of data. In this sense, data engineering is a secondary profession. I also would like to see data engineering distinguished from information engineering.

BALANCING CONFLICTING GOALS

To the Editor:

I always enjoy The Profession columns by Neville Holmes, often because I disagree with them.

The November column ("The Profession and the World," pp. 116, 114-115) led me to consider how conflicting goals complicate the attempt to apply systems analysis to public policy issues.

The only goal considered in this column is reduction of unemployment. Economists generally agree on both theoretical and empirical grounds that reducing tax rates decreases unemployment. In the public policy arena, many of those most in favor of reducing unemployment are in favor of raising—or at least not reducing—tax rates to fund an expansive government.

The balance of goals in the US and Europe has ranged widely, so there is not necessarily one obvious solution to these conflicting goals. A complete systems analysis would need to deal with issues such as a more expansive government versus a lower unemployment rate.

The issue of conflicting goals is not completely absent in the private sector, but it is more prevalent in the public policy arena. It is usually dealt with through the political process, which I believe developed specifically for this purpose.

Julian Weiss
Palatine, Ill.
jweiss27@attbi.com/

Neville Holmes responds:

It seems that this column was interpreted in ways I hadn't intended. Perhaps I should have gone from the particular to the general, rather than vice versa.

The column was intended to encourage computing professionals to "apply traditional systems analysis to the global problems we encounter." The two conflicting classes of goals I described were materialist (or economocentric) and demotic (or sociocentric), but I took pains to point out that these "two approaches need not conflict directly" and to suggest that, as global systems analysts, "we should consider material and social welfare together."

This was followed by a consideration of poverty, often depicted as the prime evil, and a suggestion that inequality is a more suitable symptom of global and national malaise to focus on. And unemployment is one obvious inequality, however it comes about, which is why I devoted some few paragraphs to the topic. However, the essay as a whole was not intended to be "a complete systems analysis" but rather a call to arms for systems analysts generally. So I would encourage readers to try for a more complete analysis than I was able to provide.

We welcome your letters. Send them to computer@computer.org. Letters are subject to editing for style, clarity, and length.

The 2003 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2003)
[illegible]
Lemor Hotel, Resort, Las Vegas, NV, USA
October 27–29, 2003

CALL FOR PAPERS: This year's conference theme pertains to the design and evaluation of intelligent interfaces for computational systems. This theme was selected to reflect the importance of human factors in systems design and use. For example, natural language can serve as an effective database interface for applications where rapid feedback is important. The IEEE International Conference on Information Reuse and Integration will feature contributed as well as invited papers. Theory, speculative, and application papers are all included in this call. The theme of the conference includes, but is not limited to, the areas listed below:

- Agent-Based Systems
- Case-Based Reasoning
- Component-Based Design
- Data Integration and Fusion
- Data Mining and Knowledge Discovery
- Decision Support Systems
- Human-Computer Interface
- Information Assurance
- Intellectual Property
- Internet Computing
- Knowledge Acquisition and Management
- Model-Based Systems
- Multimedia Systems
- Natural Language Understanding
- Quantum Computing
- Rule-Based Systems
- Soft Computing
- Software Process
- Software Reuse
- Software Testing
- Speech Recognition

ORGANIZING COMMITTEE
Honorary General Chair
Lotfi Zadeh, University of California, Berkeley, USA
General Chairs
Stuart Rubin, SPAWAR Systems Center, USA
Shu-Ching Chen, Florida Int. University, USA
Program Chair
Waleed W. Smari, University of Dayton, USA
Publications Chair
Atif Memon, Univ. of Maryland, College Park, USA
Publicity Chair
Jacob [illegible], Saint Louis University, USA
Finance Chair
David W. [illegible], Saint Louis University, USA
Local Arrangements Chair
[illegible], Western Illinois University, USA
Conference Webmaster
Andy [illegible], Saint Louis University, USA

IMPORTANT DATES

Special Sessions, Panels, and Tutorials Deadline:	Apr. 15, 2003
Full Paper Submission received by:	Jun. 1, 2003
Notification of Acceptance:	Jul. 21, 2003
Preregistration & Revised Camera-Ready Paper Due	Sep. 14, 2003

INSTRUCTIONS TO AUTHORS: Papers reporting original and unpublished research results pertaining to the above and related topics are solicited. Full paper manuscripts must be in English and should not exceed 8 pages (using the IEEE two-column template). Submissions should include the title, author(s), affiliation(s), e-mail address(es), tel/fax numbers, abstract, and postal address(es) on the first page. Papers should be submitted at the conference web site. If web submission is not possible, manuscripts should be sent as an attachment via email to the Program Chair listed below on or before the deadline date of June 1, 2003.

Dr. Waleed W. Smari,
Department of Electrical and Computer Engineering,
University of Dayton, 300 College Park,
Dayton, OH [illegible], USA
E-mail: [illegible]
Telephone (937) [illegible]; Fax (937) [illegible]

Please attach both .pdf and source file(s) (e.g., word.doc) for each submission. The subject of the email must be "IEEE IRI 2003 Submission." Papers will be selected based on their originality, timeliness, significance, relevance, and clarity of presentation. Authors should certify that their papers represent substantially new work and are previously unpublished. Organizers of prospective special sessions, panels, and tutorials are invited to submit proposals and should contact the Program Chair directly as soon as possible, but no later than April 15, 2003. Submission implies the intent of at least one of the authors to register and present the paper, if accepted. Authors of selected papers, presented at the conference, will be invited to submit an expanded version of their paper for review for possible inclusion in the appropriate IEEE SMC Transactions.

Sponsored by the IEEE Systems, Man and Cybernetics Society in cooperation with Saint Louis University