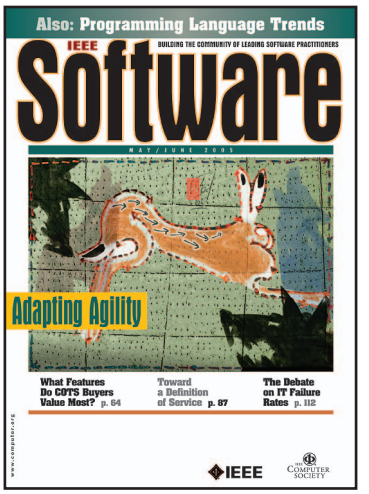
Send letters to Editor ■ software@computer.org ■ fax +1 714 821 4010

Managing Uncertainty

Just a short note in response to Todd Little's article, "Context-Adaptive Agility: Managing Complexity and Uncertainty" (May/June 2005). I found your software development project characterizations sufficiently simple and powerful to be useful. (I have long maintained that too many articles compare apples to oranges).

But I was surprised to read on page 33 that after grouping desired features into A, B, and C categories, you "... allow for uncertainty by planning the schedule so that A features consume no more than 50 percent of the overall planned effort."



At first glance, this seems to suggest that your company is still struggling to better estimate effort associated with A features, so 50 percent gives you room to maneuver (much like Microsoft apparently adds buffer time to schedules to cope with problems). On the other hand, perhaps this is simply your company's way of ensuring that there is always lots of time for B features.

So, I'm writing to ask for some additional clarification. In particular, could you provide some indication of the relative distribution of A versus B versus C feature categories and some indication of the relative distribution of A, B, and C category features that are actually implemented?

Also, here at Simlog, our project scheduling also takes into account who does what, because not all tasks are equally complex and senior staff must take on the harder ones. Of course, problems can arise when senior people are also making key contributions to other projects, so we add a load factor. This reminds the schedule owner that, for example, Bill can only be available 50 percent of the time, so if he needs to log four weeks of effort, that will run over eight calendar weeks. Are you doing something similar at Landmark Graphics?

Paul Freedman
President and cofounder, Simlog
paul.freedman@simlog.com

Todd Little responds:

Your question is a very good one. The primary reasons that we restrict A features to no more than 50 percent of the planned schedule are that we want to retain flexibility and be certain that all the A features really are mandatory. We're establishing a guarantee with product management that we'll be able to deliver all the A items by the specified date. We also plan to deliver all the B items, although we don't guarantee their delivery. We might be able to deliver some of the C items, but none are planned.

On most projects, we have roughly 25 percent A items, 25 percent B items, and 50 percent C items. On a recent set of projects, we shipped all the A and B items and 65 percent of the C items. In addition, 25 percent of the items we shipped weren't on the original list at all. We discovered these during the project and determined that they had higher priority than some C items. Some B and C items turned out

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.

0740-7459/05/$20.00 © 2005 IEEE

to be A items, necessary for the project to be commercially viable. This particular example ended up shipping a high percentage of C items, primarily because we extended the ship date, deeming it more important to add some additional functionality than to ship on a fixed date.

Since we base the plan on the A and B items, a good rule of thumb would be to expect to ship all the A items, about 50 percent of the B items, and 12.5 percent of the C items (that upon further review turn out to be As or Bs), and then expect to discover an additional 12.5 percent of the features during the project. One could expect that high-uncertainty projects would have a different distribution from low-uncertainty ones. We haven't determined that there's enough benefit to making this process more complicated. One reason we like the ABC approach is that it's easy to implement and easy for the team and stakeholders to understand.

With regard to individual scheduling, we often encounter situations where key individuals' availability turns out to be critical. In the early planning stages, we identify required skills and look at a gross-level pipeline to ensure that we can meet our guarantee to deliver the A features.

A-B-A's Limits

I was interested to read Warren Harrison's article, "Skinner Wasn't a Software Engineer," in the May/June 2005 issue about software engineering research and how to go about it differently (and cost effectively).

However, I have my doubts as to how the A-B-A technique really manages to isolate a single factor. As you mentioned in your original position paper at the ICSE 2000 workshop, skills and techniques can't be unlearned. Most tools I know of provide a program that supports a technique. Using the tool means you're exposed to the technique. For instance, providing a tool to automatically refactor code means that a subject will be exposed to refactoring as a technique and will probably continue to use it even if the tool is removed. In the case of the debugger example, I can imagine a scenario in which a naïve programmer is given a symbolic debugger. Using the tool, the programmer will be exposed to techniques such as setting breakpoints and watching variables and will develop debugging strategies to use those techniques. Remove the tool, and the programmer will probably still continue to use the same strategies (for example, by inserting print and conditional statements into the code).

Perhaps isolating the technique from the tool would be more appropriate in A-B-A experiments. In that case, the first run would be to see if the technique makes a difference (A-B). The subject would then be trained in the technique so that an A-B-A couldn't be performed. Then you'd introduce the tool, followed by removing it (A-B-C-B). In this way, only improvement due to the tool support could be demonstrated—which is a pity, as I'm sure many people (as I am) are more interested in which techniques work and which don't rather than evaluating tool support.

Andrew Doble
Chief system architect,
CSC Computer Sciences Corp.
adoble@csc.com

Warren Harrison responds:

You've hit on a weakness of the A-B-A approach. There's no way to unlearn knowledge. So, you're absolutely right in that you have to either separate the strategies from the infrastructure or make the strategies difficult or unpleasant to perform during the withdrawal phase without the infrastructure.

Of course, I suppose if you're dealing with a technique that's so obviously superior that, after getting some exposure to it, an experienced subject continues to use the concepts even without infrastructure support, maybe you don't need to go through a withdrawal period to tell that there's some therapeutic benefit to it.

So, to take your refactoring example, if an experienced subject (as opposed to a novice who really doesn't have a base of experience to draw upon) is exposed to refactoring and is so impressed with its benefits that he or she continues to use it even with the increased overhead of doing it without any tool support, we've probably learned all we'd hope to get out of such a study anyway.

We also have to be careful to consider $n = 1$ experiments as one part of a broader picture of validation. A few controlled experiments or industrial field studies aren't conclusive by themselves. Neither are $n = 1$ studies, whether they're able to complete the full A-B-A cycle or we get to B and find out this technique is so superior that once someone's exposed to it, they don't want to go back to the status quo.

I very much enjoyed Warren Harrison's editorial on software engineering experimentation ("Skinner Wasn't a Software Engineer," May/June). I agree entirely that practitioners are skeptical of results acquired from small experiments with students. Harrison writes that "finding 100 developers willing to participate in such an experiment is neither cheap nor easy. … But even if a researcher has the money, where do they find that many programmers?"

I would like to share my experiences with these challenges. First, finding the money to fund larger experiments with professionals is a matter of politics. I have yet to meet a researcher who has applied to funding bodies for money to pay for professionals to take part in experiments, nor have I done so. However, it's not the case that researchers never have sufficient funds to conduct such experiments. Five years ago, I was given the opportunity to form a new research department in software engineering at the Simula research laboratory in Oslo, Norway, with relatively few constraints on how I spent the money as long as I could envisage a good research outcome. I decided to allow about 25 percent of our budget to be used for experiments, mainly at the expense of employing a larger number of researchers. The funding body that so generously and farsightedly allowed this kind of budgeting was the Norwegian government. It might be that, were the benefits of conducting experiments with professionals to be given a sufficiently high profile by concentrated marketing ef-

forts on the part of researchers, other bodies would be persuaded to provide appropriate funding.

Next, where might we find 100 programmers? It would usually be difficult to find programmers employed in an in-house software development company because the management will prioritize the next release of their product. However, most IT consultancy companies don't have such constraints and are contracted to carry out various kinds of assignments. In the last five years, we've hired about 750 consultants from 50 companies for our experiments. Here are five examples (they cost between $50,000 and $230,000 in payment to the companies):

- 99 consultants from eight companies took part in a one-day experiment that compared two different object-oriented control styles.
- 196 consultants from 23 companies in Norway, Sweden, and the UK took part in a one-day experiment that tested the effects of pair programming.
- 20 programmers from 13 companies worked individually from one to two weeks in an experiment on UML.
- In a study on effort estimation and development processes, 35 companies presented bids for a Web-based system we needed. Four of them were selected to build the system independently of each other. The teams (two or three developers from each company) spent from seven to 25 person-weeks each.
- In another, similar study, 30 companies from 11 countries in Europe and Asia presented their bids. Four companies built the system, each spending from 10 to 20 person-weeks.

In my experience, it requires quite a lot of politics, organization, development of support tools and, of course, experimental design and analysis, but running relatively large experiments with professional developers is still feasible.

Dag Sjøberg
Research director,
Simula Research Laboratory
dagsj@simula.no

I read with enthusiasm your article concerning case studies in software engineering. I work in IT for a pharmaceutical company, and I've seen many points in common with what are called bioequivalence studies.

Briefly, this is how they work. We want to compare a generic drug versus the original formulation—for example, Tylenol versus a generic acetaminophen drug. The study design in that case will be

- Tylenol: Drug A and
- generic acetaminophen: Drug B.

We recruit approximately 50 subjects, each of whom must be present for two weekends. The first period, half of the group will receive Drug A; the second half, Drug B. (The drug administration procedure is randomized and subjects don't know which drug they are receiving.)

Then, the subjects must show up a second time and take the complementary drug (if a subject had Drug A, he must take Drug B). When the clinical part is done, the results are compared within subjects and the whole group (the analysis phase). This way, each subject serves as a baseline for the experiment. And it provides some data for conducting statistical analysis (such as Anova). So many variables affect the drug's distribution that this is the only way to compare a generic formulation to an original.

The only missing part of your article's experiment is a comparison between subjects. Having a study with $n = 1$ proves that Lint helped Bob reduce the error rate, but we can't conclude that Lint will help Lucy, Pat, or me. But, if we take the same approach with 20 subjects and create some basic statistics, the results will be more significant.

Eric Poirier
IT director, Algorithme Pharma
epoirier@algopharm.com

Warren Harrison responds:

As you point out, the approach to empirical validation doesn't differ too much whether you're doing software or medicine.

I'd never really thought much about testing generics. I guess in this case you're looking for a lack of change when moving from the original drug to the generic version or vice-versa. Probably the major confounding factor here is if subjects build up a tolerance to the drug, which might make the second drug appear to be less potent. But, of course, if this is a systemic outcome, you can account for it by mixing up the order within the subjects.

Many early controlled studies in software engineering used a similar approach to control for a procedural learning curve. If they had two treatments to compare (say, reading code with and without comments), half the subjects would get the code with comments first and half would get the code without comments first. This would account for subjects becoming more comfortable with the experimental process with time and therefore performing better on the second artifact.

Thanks for pointing out that a single $n = 1$ study wouldn't be adequate to validate a given treatment. What it is useful for is validating that for a given individual, the treatment has some effect (which is not really the case for most group studies, since they focus on the group, not the individual). As you perform more and more $n = 1$ studies, you'll acquire more and more evidence of the treatment's effectiveness.

IT Failure Rates

Robert Glass (May/June 2005, "IT Failure Rates—70% or 10–15%?") is right to question the industry's (and my) almost blind acceptance of the Standish Group's 1994 CHAOS Report of rampant IT project failure rates, especially since its use of percentages gives the illusion of precision to data from a survey of subjective opinions about project variables that few organizations actually measure objectively. In addition, I'm unclear whether the reported "189 percent overrun" means a project budgeted for $100 came in at $189 or $289.

On the other hand, my own informal ongoing surveys of clients and seminar participants repeatedly reaffirm that the folks in the trenches feel

the CHAOS figures are indeed representative of their organizations' experiences. Moreover, they universally report three situations that can't help but lead to CHAOS-type project problems:

- management-mandated budgets and schedules without apparent awareness of or regard for what needs to be accomplished or what it reasonably will take to accomplish it;
- considerable requirements and scope creep, regularly resulting in unplanned and generally unaccounted-for post-delivery work necessary to render the project workable; and
- quality generally being sacrificed to deadline, with testing invariably inadequately planned/designed, left until the end, and then squeezed in.

Robin F. Goldsmith, JD
President, Go Pro Management
robin@gopromanagement.com

As I read Robert Glass's article on IT failure rates, it occurred to me that what's missing is a discussion of what's meant by failure in an IT project.

In any project, IT or otherwise, you can identify four generic objectives: functionality, quality, cost, and schedule. Functionality defines the project deliverables' capabilities or performance characteristics, while quality defines the general level of excellence associated with them. Furthermore, in almost any project, you can identify an objectives hierarchy to the extent that some objectives are more important than others. However, this hierarchy can and will vary from one project to another.

Example 1: The government decrees a new taxation system and requires that the associated tax-processing software be available for the next fiscal year. The project to develop the software would have as its objectives hierarchy

1. schedule (have it ready by the stated date, no arguments),
2. functionality (it obviously must perform the required tax processing, but some of the bells and whistles can be left out for now),
3. quality (bugs in the first release will be accepted as long as there are workarounds), and
4. cost (keeping costs down is nice, but it'll cost what it's going to cost).

Example 2: An aircraft manufacturer develops flight software for a new airliner. This project's hierarchy would be

1. quality (no bugs whatsoever, if you please),
2. functionality (the software must do certain things, but we can negotiate on some of the bells and whistles),
3. schedule (keeping in step with the rest of the aircraft development is a good idea, but if any software problems exist, it's better to take the time to solve them than have the pilots discover them), and
4. cost (as for Example 1).

I'm sure you can think of other examples.

Project failure can be defined as failing to meet one or more objectives at the top of the hierarchy, whereas failing to meet objectives at the lower end is generally only a nuisance. I suspect that the Standish report has defined *project failure* as failure to meet any of the objectives, whereas most people involved in real IT projects would define it as failure to meet only the top one or two objectives.

Roger Graves
President, Davion Systems
rgraves@davion.com

State of the Practice

I read with great interest Robert Glass's article ("A Sad SAC Story about the State of the Practice") in the July/Aug. issue. I too, have, for many years, looked for sources of information about the state of the overall practice of software development and maintenance (including software engineering, management, and so on).

As I read the article, I couldn't help but think that while many practitioners don't participate because of alienation by the academic community, this lack of participation may additionally be a reflection of the current business environment in which software development and IT in general are practiced.

Today, the commercial climate is defined by an overriding mandate from management to do more with less and operate more efficiently. Too often, this translates into longer hours, 24/7 on-call status with the omnipresent cell phone and beeper, vacations that really don't let you get away because there's no true backup person (we'll train someone later, but later never comes), minimal (re)training opportunities (you're too important, we can't let you take two days off for training off-site—just learn as you go because no one here can backstop you), and too little time to deal with our personal (but professionally related) interests.

Add to this the time demands of a family, house, car, pets, and so on, which, we'd like to believe, bring some measure of personal satisfaction, and you see quickly that something has to give. When choosing between spending time with my children and writing or reviewing a conference paper, I'm always going to choose my family—and many others probably feel this way too. Therefore, you have a shortage of willing talent to draw from for conference purposes.

At that point, the conference organizers, optimistically hoping to draw this talent and failing, must go forward with the advertised agenda drawing on what's available: the academics. After those few in the practicing community who try to care on top of all the other burdens see that the conference proceedings are loaded with minimally relevant theoretical papers, they write the conference off as another academic kaffeeklatsch (sorry if that offends you).

Rick Kramer
Developer, P/K Solutions
rkramer@solve.net

Commercial Off-the-Shelf Confusion

I enjoyed Warren Harrison's column "The Saboteur Within" in the July/Aug. issue. Dealing with the irrational beings that we are is actually what makes a manager's job fun.

I had an issue with the rest of the magazine. Not politics, greed, or anger, but a minor annoyance. An acronym that sounds like my native Dutch word for vomit blared at me from all over the pages. The resemblance of COTS to kots doesn't bother me half as much as the excessive frequency of its use, which made me feel stupid for having to look up its meaning. Did you actively delete all occurrences of the phrase "commercial off-the-shelf," or did it simply not occur to any of the many authors to include it at least once? I guess I'm from the old school, trained to reserve repetition for emphasis, as in Professor Strunk's "omit needless words, omit needless words." And yes, we also were trained to explain acronyms at least once.

Rob Spruit
spruit@acm.org

Warren Harrison responds:

This is a new one! I've never heard anyone mention this before (interestingly, we have at least one Dutch member of our advisory board). Clearly, "COTS" does sound a lot like "kotsen." I don't know if we can get the community to change its acronyms, but we'll try to be more sensitive in the future.

This reminds me of the urban legend about GM's failure to market the Chevy Nova in Mexico since "nova" is distressingly similar to "no va," which can be loosely translated as "it doesn't go."

Roel Wieringa (Dutch advisory board member) responds:

In a multilingual world, we're bound to have some unintended puns once in a while. There used to be a conference called DOOD (Deductive and Object-Oriented Databases), meaning "death" in Dutch, and recently an IEEE workshop on Requirements Engineering Education and Training has been started, abbreviated REET, which I won't translate for you. But you might look it up in a Dutch dictionary. We have been used to the word "COTS" for some time now.

More serious, perhaps, is that people tend to forget what an acronym stands for. I regularly hear people talking about *common off-the-shelf* software. So it's good to remind the community once in a while what an acronym stands for.