Market, Don’t Propagandize

The propaganda methods you describe (From the Editor, Sept./Oct. 2004) are considered logical fallacies. As for the bandwagon, hasn’t your mother ever asked, “And if ‘everyone’ jumped off a cliff, would you follow them?” As for obtaining disapproval (also known as guilt by association), just because something is done or believed by “groups hated, feared, or held in contempt,” that doesn’t make it wrong; even a stopped clock is right twice a day. Glittering generalities are also known as “spin,” and we’re hearing plenty of examples of this in the run-up to the US elections this November.

The fact that propaganda uses logical fallacies indicates its greatest danger: you can use it to propagate false or, as history shows, evil actions and beliefs. If you can use these techniques to promote positive changes (such as structured programming), you can also use them to promote negative changes—such as reverting to the poorest of the poor-quality processes, using the hack-and-debug development approach, and monkey testing.

As a Certified Software Development Professional, I subscribe to the Software Engineering Code of Ethics and Professional Practice, which doesn’t support using logical fallacies—even for good and noble objectives; the ends do not justify the means. Granted, the ends may be in the public interest (Principle 1) or “in the best interests of the client and employer” (Principle 2), but the product—the method of convincing people to follow a course of action—wouldn’t “meet the highest professional standards possible” (Principle 3). Willfully using logical fallacies is a violation of your “integrity … in professional judgment” (Principle 4) and isn’t “an ethical approach” for a leader in the profession to take (Principle 5). If you use logical fallacies and are found out, it harms “the integrity and reputation of the profession” (Principle 6). Using logical fallacies to convince your colleagues to follow a course of action is neither “fair to” nor “supportive of” them (Principle 7). I won’t use logical fallacies but will market ideas with reason and logic, because to do otherwise doesn’t “promote an ethical approach to the practice of the profession” (Principle 8).

Propaganda has gained a negative connotation in the same way that marketing has, and for good reason. People have misused both to promote wrong and evil ideas and actions, and both have used means whose ends can’t justify them.

Norman Hines, CSDP
Software developer
Jacobs Sverdrup Naval Systems Group
norman.hines@svtnsg.com

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.

The Separation Principle Revisited

In “The Separation Principle: A Programming Paradigm” (Mar/Apr. 2004), Yasushi Kambayashi and I referred to William Cave, developer of the Separation Principle, which we described and used in the article. After
reading the article, Mr. Cave contacted me to point out misperceptions that might be inferred by readers who are unfamiliar with VisiSoft, the framework within which the Separation Principle was developed.

The C language and its derivatives—C++, Java, and so on—were never designed with the Separation Principle in mind. In fact, separation of data from instructions tends to violate object-oriented programming’s basic underpinnings. Anyone with experience using VisiSoft, which isn’t at all a C-like language, will attest to dramatic improvements in design, construction, and maintenance of real-world software.

As Yasushi’s thesis advisor, I had hoped to use VisiSoft in the comparisons we made. Unfortunately, time didn’t permit that. I’m currently working to create a larger experimental base for deriving productivity measures (for example, see “Measuring Productivity in the Software Industry,” Comm. ACM, Nov. 2003). In the meantime, I owe this letter to those people who have worked most conscientiously to develop what I consider the real silver bullet in software.

Henry Ledgard
Univ. of Toledo
hledgard@eng.utoledo.edu

In “The Separation Principle: A Programming Paradigm” (Mar./Apr. 2004), Yasushi Kambayashi and Henry F. Ledgard briefly discuss, near the end of the article, the results of an experiment they conducted. They state as their purpose

To see the Separation Principle’s effectiveness as a programming paradigm, we conducted a controlled experiment comparing the understandability of programs written using the Separation Principle with those using OOP style. Our results aren’t really conclusive but are definitely suggestive.

There’s a problem with the authors’ interpretation of the results of the two statistical hypothesis tests performed. To understand what’s wrong, let’s examine the results reported from the first test.

The authors produced two “semantically identical” programs, one written in object-oriented programming style and one written in Separation Principle style, and chose two groups of computer science and engineering majors for the study. They gave one group the OOP program and the other the SP program, and they gave both an identical set of questions to measure each student’s understanding of the program in question. The average number of correct answers from each group were compared using a one-tailed t-test. The authors concluded that they could “reject the null hypothesis that the programming paradigm in which a program is written had no impact on the number of correct answers (with 95 percent certainty).”

The claim “with 95 percent certainty”
is erroneous. The problem here is twofold. The authors fail to interpret the choice of $\alpha$ in a hypothesis test as a conditional probability, and they confuse $\alpha$ in a hypothesis test with the choice of $\alpha$ in a $(1 - \alpha)$ confidence interval. In this hypothesis test, the authors chose $\alpha = 0.05$, resulting in $1.676$ as the critical value of $t$. Let $H_0$ represent the condition that the null hypothesis is true. In truth, $\alpha = p(t > 1.676 | H_0)$—that is, $\alpha$ is the probability that $t$’s calculated value in the hypothesis test is greater than $1.676$ when $H_0$ is true. The authors interpret $\alpha$ as simply $p(t > 1.676)$. They also interpret $1 - \alpha$ as $p(t < 1.676 | H_0)$.

The choice of $\alpha$ in a hypothesis test only has meaning if the null hypothesis is actually true. When $\alpha = 0.05$, it tells us that if we perform the hypothesis test numerous times on numerous samples taken from a population in which the null hypothesis is true, then we’ll probably reject the true null hypothesis incorrectly 5 percent of the time.

When conducting a hypothesis test, we decide ahead of time how much we’re willing to risk rejecting a true null hypothesis. Choosing a smaller value for $\alpha$ lowers the risk, and choosing a larger value increases the risk. However, once we decide to reject or accept the null hypothesis, the value chosen for $\alpha$ doesn’t give us any indication whether we’ve made the right decision.

Although it would be very useful to have, the world of statistics doesn’t supply us with a measure of confidence about how right a decision is to accept or reject a null hypothesis. 

Kevin Gallagher  
kevin.gallagher@ieee.org

Correction

An error appeared in Figure 5 of “Text-Driven Modeling for Model-Driven Development” (Sept./Oct. 2004, p. 80). A connection appeared between the “idle” and “jr” boxes instead of between “in_map” and “jr.” We’ve reprinted the corrected version here.

Figure 5. Part of the state-transition chart for the active object ssm in the demo model.