Antipatterns in the Creation of Intelligent Systems

Phil Laplante, *Pennsylvania State University*
Robert R. Hoffman, *Institute for Human and Machine Cognition*
Gary Klein, *Klein Associates Division of ARA*

A design pattern is a named problem-solution pair that enables large-scale reuse of software architectures or their components (that is, interface designs[1]). Ideally, patterns explicitly capture expert knowledge, design trade-offs, and design rationale and make these lessons learned widely available for off-the-shelf use. They can also enhance developers' vocabulary—for example, by easing the transition to object-oriented programming.[2]

Conventionally, patterns consist of four elements: a name, the problem to be solved, the solution to the problem (often termed the *refactored* solution), and the consequences of the solution. Numerous sets of patterns (collectively known as pattern languages) exist for software design, analysis, management, and so on; a Web search on "pattern language" yields many hits.

Shortly after the notion of design patterns emerged, practitioners began discussing problem-solution pairs in which the solution did more harm than good.[3,4] These have come to be known as *antipatterns*, and they are well known in the design and management communities.

Antipattern examples

In 1998, researchers discussed three kinds of antipatterns: design, architectural, and management.[2] More recently, Phil Laplante and Colin Neill introduced 27 environmental antipatterns, which describe toxic work situations that can lead to organizational or project failure.[5] For example, the boiled frog syndrome discussed in this department a few years ago describes a situation in which an organization's members cannot perceive a slow loss of organizational expertise because of the subtlety of the changes.[6] Another example is the procurement problem, discussed in this department last year.[7]

Anyone can declare an antipattern—it's just a matter of whether others accept it. The pattern community relies on a "rule of three" before a new pattern or antipattern is generally accepted; that is, someone must have experienced and reported each pattern or antipattern (and a successful refactoring) in three separate instances (www.antipatterns.com/whatisapattern).

Many antipatterns take the form of cautionary tales about how day-to-day activities in human organizations can have serious repercussions. Examples include

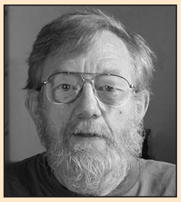
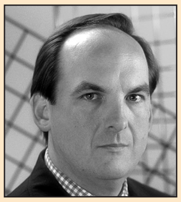
- Email Is Dangerous—we've all wished we could retrieve one we've sent; and
- Fire Drill—months of monotony followed by a crisis, then more monotony.

Several management antipatterns remind us of *Dilbert* cartoons about bad business management practices:

- Intellectual Violence—using a buzzword or arcane technology to intimidate others. Because no one really understands the technology, methodology, or practice, dismissing it is difficult.
- Blowhard Jamboree—too many industry pundits influencing technology decisions.
- Viewgraph Engineering—too much time spent building flashy presentations for customers and managers rather than working on the software.
- Death by Planning—too much planning, not enough action.
- Throw It over the Wall—management forces the latest practices or tools on the software staff without buy-in.

Other antipattern classes refer more specifically to processes in the development of information technology, especially intelligent systems. There are two reasons why we can use human-centered computing to analyze antipatterns and examine the antipattern literature to see if any design challenges or regularities of complex cognitive systems might lie therein:

- antipatterns fall at the intersection of people, technol-

Editors: Robert R. Hoffman, Patrick J. Hayes, and Kenneth M. Ford
Institute for Human and Machine Cognition, University of West Florida
rhoffman@ai.uwf.edu

Table 1. Some programming development and architecture antipatterns.

Name	Description	Negative consequence
Programming development		
Lava Flow	Dead code and forgotten design information, frozen in an overall design that is itself constantly changing	Unfixable bugs
Poltergeists	Proliferation of classes	Slower code
Spaghetti Code	Use of GOTO statements and obfuscating code structures	Code that is difficult to understand and maintain
The Blob	Procedural design leading to one giant “God” class	Code that is difficult to understand and maintain; loss of object-oriented advantages
Functional Decomposition	Structural programming in an OO language	Loss of OO advantages
Golden Hammer	Using the same design over and over again (“If all you have is a hammer, everything is a thumb.”)	Suboptimal designs
Metric Abuse	Naive or malicious use of metrics (for example, using the wrong metric or choosing a metric to get back at someone)	Unanticipated consequences; inability to control the project; fear, uncertainty, and doubt
Road to Nowhere	Regarding plans and requirements as rigid roadmaps; development teams failing to adapt	Creating superficial plans such as Gantt charts and PERT (Program Evaluation and Review Technique) charts to depict event sequences as if plans are actually followed; as artifacts, plans aren’t a substitute for the evolving activity itself
Architecture		
Name	**Description**	**Negative consequence**
Architecture by Implication	Lack of architectural specifications for a system under development	Difficulties in maintaining and extending the system
Design by Committee	Everything but the kitchen sink	Political or policy factors dominating technical issues
Reinvent the Wheel	Failure to use acceptable existing solutions	Increasing cost and time to delivery
Stovepipe System	Legacy software with undesirable qualities	Difficulty in understanding and maintaining the system
Vendor Lock-In	Captive system(s) with mandated legacy components	Higher overall cost, loss of flexibility, and systems hostile to users
Penny Foolish	Focusing on short-term cost considerations	Burden is placed on the users, who have to kluge around hostile software.

ogy, and cognitive work (including the work of creating new intelligent technologies); and
- antipatterns’ negative consequences tend to involve making systems toxic (that is, unusable) for humans.

Table 1 presents two classes, programming development antipatterns and architecture antipatterns.

Two new classes

Here, we add two new classes to the roster: procurement antipatterns and unintelligent-systems antipatterns (see tables 2 and 3). Many of these derive from recent experience in cognitive systems engineering.

Procurement antipatterns

One procurement antipattern we see far too often is Buzzword Mania. Sponsors of research and development programs ask for the world, and providers gladly promise it, using jargon-laden phrases that no one really understands, such as the following: “[insert acronym here] will provide near-real-time interoperability using a robust framework leveraging a multimatrix solution to inference that will use adaptive configuration management to ensure ….”

Note the widespread use of the word “will” when in fact the proposer should use words such as “might” or “could,” or even more honestly, “we hope will.” Alas, we have no clear cases of successful refactoring to point to, but we are hopeful that some are out there.

Unintelligent-systems antipatterns

Unintelligent systems purport to provide “intelligent” functionality through some combination of hardware and software but are really created by relying on designer-centered design rather than on the empirical realities of cognitive work and sociotechnical organizations. Such systems require significant numbers of workarounds and kluges; at worst, they’re useless.

Unintelligent-systems antipatterns differ from poor design—they are ways in which machines actually make people dumb. We’ve seen the consequences of unintelligent-systems antipatterns many times. Here are just two examples:

- The FBI spent a lot of money to develop the Trilogy system as a part of an information technology modernization program. The resulting software doesn’t support the analysts’ “operational needs”—that is, the actual cognitive work they do.[8]
- In the aftermath of the destruction of the World Trade Towers, massive amounts of funding have gone into creating new software systems to aid intelligence analysts’ work. A new generation of technologists was led to believe that analysts need systems to help them overcome human limitations. This opinion was based on an older psychology theory[9] that describes cognition in terms of dozens of reasoning biases, such as the

Table 2. Five procurement antipatterns (adapted from W.H. Brown et al.[2]).

Name	Description	Negative Consequence
Fools Rush In	Rushing to use a new methodology, tool, or platform. The IT and software development world is ever-changing, and new ideas, technologies, methodologies, and practices arise all the time. Being one of the earliest adopters is often unwise.	Usually the "latest and greatest" is based more on hype than trusted evidence—more "sizzle" than "steak."
Emperor's New Clothes (also known as Smoke and Mirrors)	Those who perceive instances of Buzzword Mania are inhibited from speaking up. No one wants to point out the obvious, embarrassing truth about a situation or claim. It's often convenient to lay our hopes on a technology or methodology about which we know little, thereby providing plausible hope for a miracle.	Overly aggressive use of demonstration systems for sales purposes. The sponsor is sold a bill of goods. The truth emerges, finally when users start using the new software.
Metric Madness (also known as John Henry and the Seam Hammer)	Evaluating software systems by measuring easily measured things (such as efficiency, accuracy, and raw productivity) and generally regarding the human user as an output device.	Failure to look at things that are hard to measure but critical (such as accelerating the achievement of expertise, facilitating problem recognition, coping with goal trade-offs, and reinforcing intrinsic motivation)
The Rolling Stone (also known as I Can't Get No Satisfaction)	Evaluation by satisficing (some people like it, more or less, some of the time) based on one-off and superficial interviews, with little awareness of issues of interviewing (such as biasing) or the effect of task demand characteristics. Is someone who has invested time and effort in building a system really going to fess up and say "It's lousy"?	Failure to objectively, empirically investigate usability and usefulness issues.
Potemkin Village (Around 1772, Potemkin built a façade village to fool Catherine the Great into thinking that all was well in the poverty-stricken Ukraine.)	A fancy but superficial facade hides substantial defects or shortcomings. People reach for a solution before the problem is even fully described. Many tool sets, frameworks, and off-the-shelf products allow for the quick manufacture of sophisticated-looking solutions but fail to promote robust designs and belie proper testing throughout the life cycle.	Highly staged demos and presentations; reluctance to let the customer "look under the hood;" systems that don't do anything useful and end up collecting dust.

disconfirmation bias or the base rate fallacy.[10] The new software systems encourage (we might say *force*) analysts to juggle probabilities and evaluate likelihoods, all for the sake of shoehorning reasoning into a "rational" mode, such as Bayesian inference. Word on the street seems to be that analysts simply don't like the tools. The tools have little apparent value-added for the analyst and create what is perceived to be a considerable amount of make-work.[11]

We present three salient unintelligent-systems antipatterns, which we've adapted from Gary Klein's work.[12]

The Man behind the Curtain (from the Wizard of Oz). Information technology usually doesn't let people see how it reasons; it's not understandable.[13] If human users can't understand and their easiest path is to accept the machine's judgment, they are opted out of opportunities to practice their own mental model formation and evaluation skills. We aren't suggesting that all users must always be able to see how code is executing. But we've seen many cases where algorithms' mysteries nullified the potential benefits from human reasoning and collaboration:

> The [US] Air Force developed [AI] programs to generate Air Tasking Orders. Ordinarily, generating ATOs takes days ... the program could produce one in just hours .... The Air Force planners, however, did not like the new system .... [When we investigated how the planners made ATOs] we did not observe anyone evaluating an ATO once it was finished, but we did see planners evaluating the ATO while it was being prepared. One planner would suggest an approach, others would debate the suggestion, perhaps make some improvements. When the ATO was finished, the planning team would have a thorough appreciation for the nuances …. But when the [AI] software produced the ATO, the users lost their opportunity to conduct an evaluation, with no way to appreciate the rationale behind the order. (p. 263)[12]

Hide-and-Seek. On the belief that decision-aids must transform data into information and information into knowledge, data are actually hidden from the decision maker. The negative consequence of this antipattern is that decision makers can't use their expertise. The filtering and transforming of data, for the sake of protecting the decision maker from being overwhelmed, prevents the decision maker from drilling down and forces him or her to rely on other people's (or machines') decisions. On the belief that computers can miraculously present the decision maker with "the right information at the right time," data are (in one way or another) filtered into essential versus nonessential categories (on the basis of the judgments of the software designer—the person behind the curtain). The decision maker sees only the information that the computer (as a stand-in for the designer) thinks are important. Ideally this filtering is at least done in a smart, context-sensitive way, but however it is done, there is always a risk of turning humans into passive recipients. Experts like to be able to control their own searching and learning, and with good reason. We have seen many instances where experts did not believe the massaged data, went back to search the raw data, and made decisions—correct decisions—that differed from those coughed up by the computer, or by other decision makers who weren't skeptical and didn't do their own drill-down.

> One of the expert weather forecasters we followed complained bitterly about a new computer system that had replaced his old machine. The new system could show trends and plot curves and do all kinds of computer tricks. And that was the problem. If the temperatures in a region were very variable, some high and some low, the computer would smooth these out to provide a uniform temperature curve for the region. The program developers had wanted to provide operators with a sense of the trends, and to do that they had filtered out the "noise." But the expert had always depended on seeing

Table 3. An unintelligent-systems antipattern.

Name	What happens	Negative consequence
Shoeless Children (The shoemaker is too busy earning a living to make shoes for his own children.)	In some cases, systems engineers conduct what we might call cognitive task analysis, but they use weak methods (such as interviewing) that are not rich and continuous throughout project development. In other cases, where cognitive systems engineers are involved, they aren't given sufficient time or resources to use their expertise. In both cases, the result is that the development team is deprived of the use of and potential benefit from the tools needed to do the job right, usually in the guise of conservation.	Systems that aren't usable, useful, or understandable; systems that force users into a make-work mode far too much of the time and require significant numbers of kluges and workarounds.

> these areas of turbulence. To him, they signaled some sort of instability. Whenever he saw this cue, it triggered a reaction to watch these fronts much more carefully because it was a sign that something was brewing. As this forecaster said, "In reality, the fronts 'wiggle' as they move across the land. And it's in those whorls and wiggles that weather happens." The new system had erased this cue, making him less sensitive to the way the weather was developing and hurting his ability to do his job. (p. 252)12

As we suggested in the essay on sense-making in this department,[14] only human minds can drill down to find out what the right cues are in the first place, especially in non prototypical situations.

The Mind Is a Muscle. In the attempt to acknowledge human factors in the procurement process, some guidelines end up actually working against human-centering considerations: "Design efforts shall minimize or eliminate system characteristics that require excessive cognitive, physical, or sensory skills."[15] We find this astounding—that information systems should, in effect, prevent people from working hard and thereby progressing along the continuum of proficiency. Ample psychological evidence shows that we achieve expertise only after lots of "deliberative practice," in which we're intrinsically motivated to work hard and work on hard problems.[16]

> Information technology can diminish the active stance found in intuitive decision makers, and transform them into passive system operators. Information technology makes us afraid to use our intuition; it slows our rate of learning … passivity is bad enough but it can degenerate to the point where decision makers assume that the computer knows best … they follow the system's recommendations even in those cases where their own judgment is better than the system's solution. (p. 265)[12]

Recognizing unintelligent-systems antipatterns and giving them names might lead to system remediation more quickly, or at least the chance that a repository of lessons learned might be put to some use. And the advantages of avoiding unintelligent-systems antipatterns altogether is clear—successful system delivery, reduced costs in the long run, and reduced user frustration. In the spirit of what we might call the Penny Foolish antipattern (see table 1), reduced costs must include the human costs that are incurred after systems are "delivered": the costs to users in terms of frustration, the costs to human resources in terms of retraining personnel. Instead of computing the *total cost of ownership*, sponsors should calculate the *total human cost of ownership* by taking into account factors such as (re)training costs, costs of worker turn-around, costs of loss of expertise, and so on.

How can unintelligent-systems antipatterns be refactored? In some cases, unfortunately, refactoring involves throwing away or completely redesigning the system. In some cases, we can perhaps, just perhaps, repurpose the system's intent. Only by conducting cognitive task analysis in the first place can we have a path to creating human-centered systems that are useful, usable, and understandable and that help in growing expertise and maturing the cognitive work. This is true for many of the antipatterns we have mentioned in this essay: Only by doing it right the first time do we have a chance at a solution. Thus, we conclude with one unintelligent-systems antipattern, called Shoeless Children (see table 3).

In IT and software development, commentators and luminaries have long advocated the pursuit of excellence and the application of best practices. In recent years, however, what we have seen is a weakened, tacit view that "best is the enemy of good enough." Perhaps we misconstrue the true sentiment, but to suggest that computer scientists provide or offer anything but their best in any situation seems alien to most people. We would accept "best, time permitting" as a compromise, but never that the best is the enemy of good enough, or that the goal, especially for intelligent systems, is to be merely good enough. ■

References

1. T. Stanard et al., "HCI Design Patterns for C2: A Vision for a DoD Reference Library," *State-of-the-Art Report*, Air Force Research Laboratory, Wright-Patterson Air Force Base, 2006.
2. W.H. Brown et al., *Anti Patterns: Refactoring Software Architectures and Projects in Crisis*, John Wiley & Sons, 1998.
3. C.A. Dekkers and P.A. McQuaid, "The Dangers of Using Software Metrics To (Mis)Manage," *IT Professional*, Mar./Apr. 2002, pp. 24–30.
4. T. DeMarco, *Why Does Software Cost So Much?* Dorset House Publishing, 1995.
5. P.A. Laplante and C.J. Neill, *Antipatterns: Identification, Refactoring, and Management*, Auerbach Press, 2006.
6. R.R. Hoffman and L.F. Hanes, "The Boiled Frog Problem," *IEEE Intelligent Systems*, July/Aug. 2003, pp. 2–5.
7. R.R. Hoffman and W.C. Elm, "HCC Implications for the Procurement Process," *IEEE Intelligent Systems*, Jan./Feb. 2006, pp. 74–81.
8. J.C. McGraddy and H.S. Lin, eds., *A Review of the FBI's Trilogy Information Technology Modernization Program*, tech. report, Computer Science and Telecommunications Board Div. on Eng. and Physical Sciences, Nat'l Academies Press, 2004.
9. R. Heuer, *Psychology of Intelligence Analysis*, Central Intelligence Agency, 1999.
10. J. Flach and R.R. Hoffman, "The Limitations of Limitations," *IEEE Intelligent Systems*, Jan./Feb. 2003, pp. 94–97.
11. B. Moon, A.J. Pino, and C.A. Hedberg, "Studying Transformation: The Use of Cmap-Tools in Surveying the Integration of Intelligence and Operations," *Proc. 2nd Int'l Conf. Concept Mapping*, 2006, pp. 527–533; available from proteaed@cariari.ucr.cr.
12. G. Klein, *Intuition at Work*, Doubleday, 2003.
13. D.D. Woods and E. Hollnagel, *Joint Cognitive Systems: Patterns in Cognitive Systems Engineering*, CRC Press, 2006.

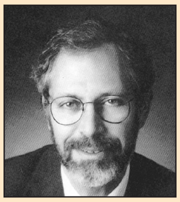
14. G. Klein, B. Moon, and R.R. Hoffman, "Making Sense of Sensemaking 1: Alternative Perspectives," *IEEE Intelligent* Systems, July/Aug. 2006, pp. 22–26.

15. *Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information Systems Acquisition Programs*, instruction 5000.2-R, US Dept. of Defense, 1996, paragraph C5.2.3.5.9.1.

16. K.A. Ericsson, R.Th. Krampe, and C. Tesch-Römer, "The Role of Deliberate Practice in the Acquisition of Expert Performance," *Psychological Rev.,* vol. 100, 1993, pp. 363–406.

Phil Laplante is a professor of software engineering at Pennsylvania State University. Contact him at plaplante@psu.edu.

Robert R. Hoffman is a senior research scientist at the Institute for Human and Machine Cognition. Contact him at rhoffman@ihmc.us.



Gary Klein is chief scientist in the Klein Associates Division of Applied Research Associates. Contact him at gary@decisionmaking.com.

IEEE Intelligent Systems

2007

EDITORIAL CALENDAR

January/February: *AI's Cutting Edge*
This issue will examine a variety of topics at the forefront of AI research, including natural language processing, data mining, planning systems, intelligent transportation systems, and evolutionary computing.

March/April: *Interacting with Autonomy*
For at least the foreseeable future, people will still need to interact with autonomous systems at various levels of involvement as conditions change dynamically. This special issue will present articles on human interaction with autonomous or semiautonomous physical systems such as ground-based robots, unmanned aerial vehicles, and assistive technologies.

May/June: *Recommender Systems*
Recommendation problems have a long history as a successful AI application area. The interest in such systems has dramatically increased owing to the demand for personalization technologies by large, successful e-commerce applications. This special issue will report on successful implementations and on future research directions in recommender technologies.

July/August: *Intelligent Educational Systems*
The distributed nature of 21st-century education poses strong demands for intelligent educational systems tailored to students' and teachers' individual needs. This special issue will discuss novel methods, tools, and applications addressing this field's key challenges, such as semantic interoperability, context-sensitive feedback generation, and personalized content delivery and generation.

September/October: *Social Computing*
This special issue will investigate the development and use of social software—tools and computing methods that support social interaction and communication in significant application domains. It will also feature articles describing general intelligent systems (not necessarily social software) that leverage insights and findings from social, organizational, cultural, and media theory.

November/December: *Argumentation Technology*
Formal models of argumentation have been gaining increasing importance in AI and have found a wide range of applications. This special issue will report on the state of the art of AI-based argumentation applications, focusing on deployed systems or pilot systems that provide promising techniques and ideas.

Bringing You the Latest Artificial Intelligence Research
WWW.COMPUTER.ORG/INTELLIGENT