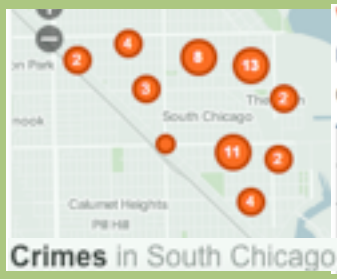
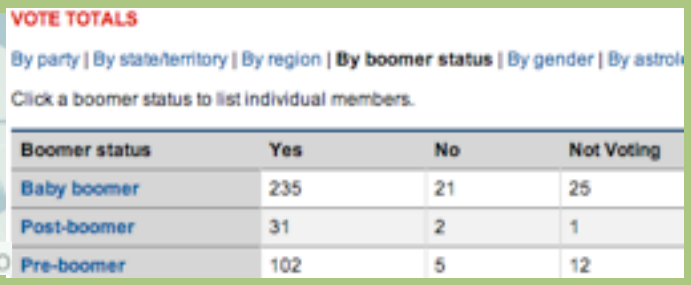
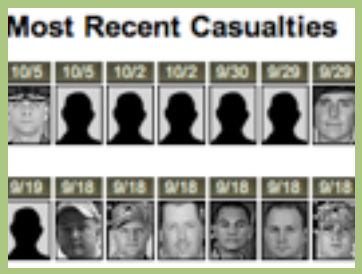
The programmer as journalist:







a Q&A with Adrian Holovaty

By Robert Niles 6/5/2006

Washingtonpost.com's Web tech guru answers questions about programming's role in news reporting and presentation

From Online Journalism Review, http://www.ojr.org/ojr/stories/060605niles

[The universe of journalists who program is, well, pretty small. Which is why I welcome the chance to talk with Adrian Holovaty, an award-winning journalist/ programmer whose work, both for WashingtonPost.com and for his own sites, expands this profession's capabilities. Adrian graciously agreed to answer a few of my questions via e-mail for OJR. -- Robert]

OJR: I think one can safely assume that everyone in the news business understands how one "does journalism" through writing or photography. But how does one "do journalism" through computer programming?

Holovaty: The way I see it, there are three basic tasks that journalists do:

1. Gathering information. This involves talking to sources, examining documents, taking photographs, etc. It's reporting.

2. Distilling information. This involves applying editorial judgment to decide what parts of the gathered information are important and relevant.

3. Presenting information. This involves shaping the distilled information into a format that is accessible to the readership. Some examples: writing style (inverted pyramid, etc.), photo color-correction, newspaper page design.

"Doing journalism through computer programming" is just a different way of accomplishing these goals. Namely, the technique favors automation wherever possible.

For example, it's possible to automate that first step, the gathering of information. That's how my chicagocrime.org site works. Each weekday, my computer program goes to the Chicago Police Department's website and gathers all crimes reported in Chicago. Similarly, the U.S. Congress votes database I helped put together at washingtonpost.com works the same way: Several times a day, an automated program checks several government websites for roll-call votes. If it finds any, it gathers the data and saves it into a database.

The second step, distilling information, can also be automated. Just as an editor can apply editorial judgment to decide which facts in a news story are most important, a programmer-journalist (we really do need a better name than that!) decides which ***queries*** should be made of data. For instance, on chicagocrime.org I decided it would be useful if site users could browse by crime type, ZIP code and city ward. On the votes database site, we decided it would be useful to browse a list of all the votes that happen late at night and a list of members of Congress who've missed the most votes. Once we made that decision of which information to display, it was just a matter of writing the programming code that automated it.

In the "journalism through computer programming" realm, the third step, presentation, is also automated. This is particularly complex, because in creating websites, it's necessary to account for all possible permutations of data. For example, on chicagocrime.org I had to account for missing data: How should the site display crimes whose data has changed? What should happen in the case where a crime's longitude/latitude coordinates aren't available? What should happen when a crime's time is listed as "Not available"?

Also, I should point out that the two example sites I've given are entirely automated, but often it's not possible to automate an entire project. In most cases, information gathering is done by humans rather than computers, and the computer programming comes into play in automating the distillation and display of the data.

A good example of this is washingtonpost.com's Faces of the Fallen site, which lists all known U.S. service members who have died in Iraq and Afghanistan. That information is collected by the Post's fantastic newsroom research team, not by automated scripts. The "journalism via computer programming" in this case is in the setup of the website itself: Once our researchers collect and verify information, it gets displayed on the website and is made browsable and searchable by a variety of different parameters such as age, home town and military branch. That -- the display -- is the part that's automated.

OJR: What is the value to a journalist in understanding programming, or even learning how to do it?

Holovaty: The main value in understanding programming is the advantage of knowing what's possible, in terms of both data analysis and data presentation. It helps one think of journalism beyond the plain (and kind of boring) format of the news story.

Programming comes in handy in all sorts of other areas, too, including gathering information. Now that quite a few governments and organizations are publishing data on their own websites, it's a valuable skill to be able to automate the retrieval of that data and compile it into a format that makes it easy to research and aggregate.

OJR: What should journalism schools be doing to prepare future journalists to work in a mash-up publishing universe?

Holovaty: J-schools need to get way more technical. A graduate of a journalism school should be a master of collecting data -- whether the old-fashioned way (by talking to humans) or through automated means.

The closest thing journalism schools currently have (to my knowledge) is computer-assisted reporting classes. Those classes should be required, in my opinion, and even better would be for j-schools to partner with computer-science departments so that journalism students would get some experience coding.

OJR: What types of information are newsrooms collecting right now, but most under-utilizing on their websites?

Holovaty: Much of the information that journalists collect, day to day, is structured. Information such as crime reports, obituaries and event listings always follow a certain pattern, which can be richly exploited by databases.

The majority of newspapers takes the time to ***collect*** this information -- which is the hard part -- but they dramatically reduce its value by

NOT storing it in structured formats. Instead, they distill it into big blobs of text for publication in their print editions, and then they shovel those big blobs of text onto their websites. At this point, all structure is lost: Crime reports can't be sorted or searched intelligently, and event listings can't be viewed in any sort of user-friendly way.

The very act of distilling information into a news story -- which is essentially a big blob of text -- removes any sort of structure. Information is exponentially more valuable if it's structured.

So I urge news companies to retain as much structure in their information as possible. These days, it's easier and cheaper than ever to set up a database server. Just do it.

A few specific examples? Any sorts of public records are structured, really. Crime reports are an obvious one. Fire-station reports, local school data, transportation data. There's a ton of this stuff.

Beyond the obvious examples, journalists should step back and consider more abstract concepts in terms of structured information. For example, just a couple of weeks ago at washingtonpost.com we databased the "key races" across the country in the 2006 elections, as determined by our editors: http://projects.washingtonpost.com/elections/keyraces/ . Each race has a name, a state/district, a number of candidates -- it's very structured, if you think about it that way. And because we've databased it, we've automated much of the tedium of updating the site, because the site runs itself, grabbing information from our database.

This sort of automation and exploitation of structured information is where I think (and hope) journalism is going.

OJR: What ought news organizations do to encourage tech innovation from their staffs?

Holovaty: Hire programmers! It all starts with the people, really. If you want innovation, hire people who are capable of it. Hire people who know what's possible.

And once you hire the programmers, give them an environment in which they can be creative. Treat them as bona fide members of the journalism team -- not as IT robots who just do what you tell them to do.

OJR: Do you think most news managers are afraid of technology? If so, how do tech-savvy journalists overcome that?

Holovaty: I've met both types of managers -- those that are scared and those that aren't. (For the news managers who ***are*** afraid of technology, you can't blame 'em. It's only natural. Technology is completely changing their industry, whose rules haven't changed drastically in a long time.)

It seems the best way to overcome the fear is to emphasize that technology can be used to further the goals of journalism. It's reasonable for managers to be afraid of things they don't understand, but if you boil down the specific technology to the specific journalism problems it solves, I suspect managers would be more understanding.

OJR: What is the most innovative project you've worked on? What was so interesting about it?

Holovaty: The projects that are most interesting to me involve reverse-engineering and altering Internet applications to do things they weren't supposed to do, for the benefit of users. For example, a year ago I tinkered with putting CTA (Chicago Transit Authority) subway maps on Google Maps (http://holovaty.com/blog/archive/2005/04/19/0216/). It no longer works, but it was really cool. Also, I enjoyed creating the "All-Music Guide Fixer" Firefox extension, which, when installed, alters the display and functionality of allmusic.com (http://holovaty.com/blog/archive/2004/07/19/2210). This idea -- site-specific user customizations of websites -- eventually became the Greasemonkey Firefox extension.

In journalism, I'd have to say the most innovative project I've been

lucky enough to work on was lawrence.com, the local entertainment site for Lawrence, Kansas. So much automated subtlety is happening behind the scenes of that site. For example, in the event calendar, an event that takes place at a bar will automatically pull out the drink specials for the day of the event. Similarly, if an event features a local band, the system automatically pulls out sound clips and creates an "If you go, you might hear these songs" sidebar. Lawrence.com has a ton of little innovations that go way beyond what most other entertainment sites do, even though the site has had these little innovations for more than three years.

OJR: What interesting projects are you working on now?

Holovaty: I'm heavily involved in the development of Django, an open-source Web framework for the Python programming language. In layperson's speak, it's free software that makes Web development fast and easy. We created it when I worked in Lawrence, and we open-sourced it in July 2005. It's gotten a ton of attention, and people all over the world are using it and improving it. I'm cowriting a book about it at the moment, as well.

Aside from that, I've been collecting various public-record data in Chicago in preparation for the launch of my "sequel" to chicagocrime.org. Can't say much more about this project at the moment, but I'm very excited to launch in the coming weeks! [This is EveryBlock.com ---KSD]

OJR: Other that the stuff you're working on, what technology you've looked at recently has grabbed your attention?

Holovaty: Generally I get excited by new APIs that various websites are launching. The Flickr APIs are a classic example: They let any programmer query the Flickr photo database via programs.

OJR: Journalism's always been a competitive business. But what technical initiatives should news organizations be cooperating on? What opportunities, if any, are the industry missing when companies don't work together?

Holovaty: I think news organizations should cooperate on removing mandatory Web-site registration walls, which are severely reader-unfriendly. It's embarrassing to be associated with an industry that treats its customers with such disdain.

OJR: What online news projects have you seen recently, if any, that you thought were especially well done? (Not counting the Washington Post and other sites you've worked on....)

Holovaty: Off the top of my head --

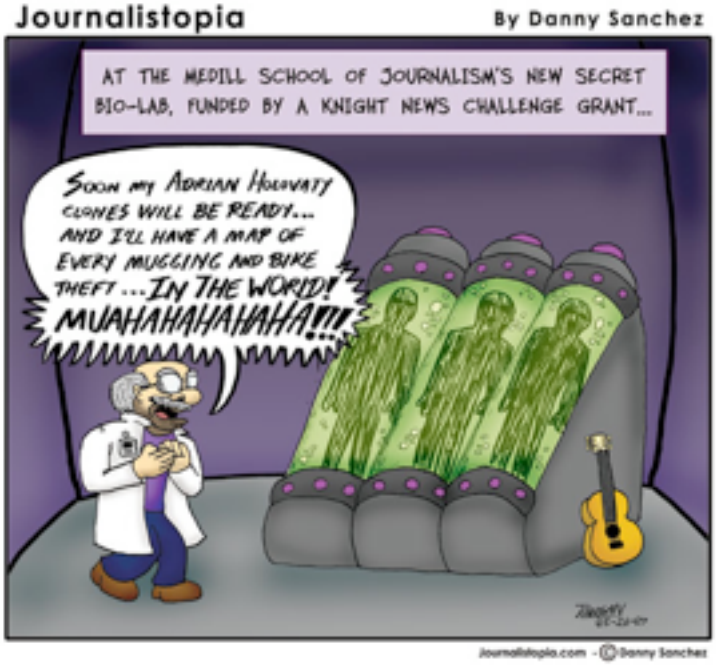
* Just the other day I saw the great weather/hurricane tracking app at http://www.ibiseye.com.

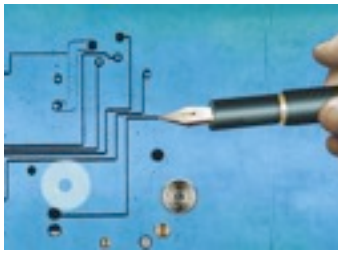
* I'm consistently impressed by the stuff coming out of mySociety .

* Faneuil Media does some great work.

OJR: What tech sites do you check to keep up with the latest in mash-ups, programming and Web development?

Holovaty: Every day I check delicious popular a couple of times. That's a good indicator of what people are talking about and the new things happening on the Web.





Decoding the Value of Computer Science

Chronicle of Higher Education, Nov 7, 2010

By Kevin Carey

In The Social Network, a computer-programming prodigy goes to Harvard and creates a technology company in his sophomore dorm. Six year later, the company is worth billions and touches one out of every 14 people on earth.

Facebook is a familiar American success story, with its founder, Mark Zuckerberg, following a path blazed by Bill Gates and others like him. But it may also become increasingly rare. Far fewer students are studying computer science in college than once did. This is a problem in more ways than one.

The signs are everywhere. This year, for the first time in decades, the College Board failed to offer high-school students the Advanced Placement AB Computer Science exam. The number of high schools teaching computer science is shrinking, and last year only about 5,000 students sat for the AB test. Two decades ago, I was one of them.

I have never held an information-technology job. Yet the more time passes, the more I understand how important that education was. Something is lost when students no longer study the working of things.

My childhood interest in programming was a product of nature and nurture. My father worked as a computer scientist, first in a university and then as a researcher for General Electric. As a kid, I tagged along to his lab on weekends, watching him connect single-board DEC computers into ring networks and two-dimensional arrays, feeling the ozone hum of closet-sized machines. By my adolescence, in the mid-1980s, we had moved to a well-off suburb whose high school could afford its own mainframe. That plus social awkwardness meant many a night plugged into a 300-baud modem, battling other 15-year-olds in rudimentary deep-space combat and accumulating treasure in Ascii-rendered dungeons without end.

Before long I wanted to understand where those games came from and how, exactly, they worked. So I took to programming, first in Basic and then Pascal. Coding taught me the shape of logic, the value of brevity, and the attention to detail that debugging requires. I learned that a beautiful program with a single misplaced semicolon is like a sports car with one piston out of line. Both are dead machines, functionally indistinguishable from junk. I learned that you are good enough to build things that do what they are supposed to do, or you are not.

I left for college intending to major in computer science. That lasted until about the fifth 8:30 a.m. Monday class, at which point my enthusiasm for parties and beer got the upper hand, and I switched to the humanities, which offered more-forgiving academic standards and course schedules. I never touched Pascal again.

Fortunately, other students had more fortitude. They helped build and sustain the IT revolution that continues to make America the center of global innovation. But the number of people going this way is in decline. According to the Computing Research Association, the annual number of bachelor's degrees awarded in computer science and computer engineering dropped 12 percent in 2009. When Zuckerberg started Facebook, in 2004, universities awarded over 20,000 computer degrees. The total fell below 10,000 last year.

This "geek shortage," as Wired magazine puts it, endangers everything from innovation and economic growth to national defense. And as I learned in my first job after graduate school, perhaps even more.

There, at the Indiana state-budget office, my role was to calculate how proposals for setting local school property-tax rates and distributing funds would play out in the state's 293 school districts. I did this by teaching myself the statistical program SAS. The syntax itself was easy, since the underlying logic wasn't far from Pascal. But the only way to simulate the state's byzantine school-financing law was to

understand every inch of it, every historical curiosity and long-embedded political compromise, to the last dollar and cent. To write code about a thing, you have to know the thing itself, absolutely.

Over time I became mildly obsessed with the care and feeding of my SAS code. I wrote and rewrote it with the aim of creating the simplest and most elegant procedure possible, one that would do its job with a minimum of space and time. It wasn't that someone was bothering me about computer resources. It just seemed like the right thing to do.

Eventually I reached the limit of how clean my code could be. Yet I was unsatisfied. Parts still seemed vestigial and wrong. I realized the problem wasn't my modest programming skills but the law itself, which had grown incrementally over the decades, each budget bringing new tweaks and procedures that were layered atop the last.

So I sat down, mostly as an intellectual exercise, to rewrite the formula from first principles. The result yielded a satisfyingly direct SAS procedure. Almost as an afterthought, I showed it to a friend who worked for the state legislature. To my surprise, she reacted with enthusiasm, and a few months later the new financing formula became law. Good public policy and good code, it turned out, go hand in hand. The law has re-accumulated some extraneous procedures in the decade since. But my basic ideas are still there, directing billions of dollars to schoolchildren using language recorded in, as state laws are officially named, the Indiana Code.

A few years later, I switched careers and began writing for magazines and publications like this one. It's difficult work. You have to say a great deal in a small amount of space. Struggling to build a whole world in 3,000 words, I thought back to stories my father would tell of consulting on the Galileo space-probe project. To make it work with 1970s technology and withstand the rigors of deep space, he and his fellow programmers had to fit every procedure —"take pictures," "fire rockets," "radio NASA back on Earth"—into a machine with 64K bytes of what we now call RAM. That's about 1/60,000th of what you can buy in a cheap laptop at Best Buy today. So every program on Galileo was polished to a gleam.

Good editors know there is no Moore's law of human cognition that would double our ability to retain and process information every 18 months. Instead, the technology-driven surfeit of modern information has made the need for clarity and concision more acute.

My editors rarely said a word about words, in the sense of how to phrase an idea. The real work was in structure, in constructing an unbroken chain of logic, evidence, and emotion that would bring readers to precisely where I wanted them to be. I learned to minimize the number of operating variables (people). I also learned that while some talented writers can get away with recursive scene-setting, it is usually best to start at the beginning and finish at the end. I discovered that skilled writers, like programmers, embed subtle pieces of documentation in their prose to remind readers where they are and where they are going to be.

Writing, in other words, is just coding by a different name. It's like constructing a program that runs in the universal human operating system of narrative, because, as Joan Didion once said, we tell ourselves stories in order to live. Authors adhere to a set of principles as structured in their own way as any computer language. Publications worth writing for insist on Galilean standards of quality and concision.

Computer science exposed two generations of young people to the rigors of logic and rhetoric that have disappeared from far too many curricula in the humanities. Those students learned to speak to the machines with which the future of humanity will be increasingly intertwined. They discovered the virtue of understanding the instructions that lie at the heart of things, of realizing the danger of misplaced semicolons, of learning to labor until what you have built is good enough to do what it is supposed to do.

I left computer science when I was 17 years old. Thankfully, it never left me.