Creativity in Computer Science

Daniel Saunders and Paul Thagard
University of Waterloo

Saunders, D., & Thagard, P. (forthcoming). Creativity in computer science. In J. C. Kaufman & J. Baer (Eds.), *Creativity across domains: Faces of the muse*. Mahwah, NJ: Lawrence Erlbaum Associates.

1. Introduction

Computer science only became established as a field in the 1950s, growing out of theoretical and practical research begun in the previous two decades. The field has exhibited immense creativity, ranging from innovative hardware such as the early mainframes to software breakthroughs such as programming languages and the Internet. Martin Gardner worried that "it would be a sad day if human beings, adjusting to the Computer Revolution, became so intellectually lazy that they lost their power of creative thinking" (Gardner, 1978, p. vi-viii). On the contrary, computers and the theory of computation have provided great opportunities for creative work.

This chapter examines several key aspects of creativity in computer science, beginning with the question of how problems arise in computer science. We then discuss the use of analogies in solving key problems in the history of computer science. Our discussion in these sections is based on historical examples, but the following sections discuss the nature of creativity using information from a contemporary source, a set of interviews with practicing computer scientists collected by the Association of Computing Machinery's on-line student magazine, *Crossroads*. We then provide a general comparison of creativity in computer science and in the natural sciences.

2. Nature and Origins of Problems in Computer Science

Computer science is closely related to both mathematics and engineering. It resembles engineering in that it is often concerned with building machines and making design decisions about complex interactive systems. Brian K. Reid wrote: "Computer science is the first engineering discipline ever in which the complexity of the objects created is limited by the skill of the creator and not limited by the strength of the raw materials" (Frenkel, 1987, p. 823). Like engineers, computer scientists draw on a collection of techniques to construct a solution to a particular problem, with the creativity consisting in development of new techniques. For example, during the creation of the first large-scale electronic computer, Eckert and Mauchly solved numerous engineering problems, resulting in solutions which became important contributions to computer science (Goldstine, 1972).

Computer science also has a strong mathematical component. An early example is Alan Turing's invention of an abstract, theoretical computing machine in 1935 to solve David Hilbert's decidability problem in the foundations of mathematics (Hodges, 1983). The Turing machine has proven to be a very powerful tool for studying the theoretical limitations of computers. The theory of the class of NP-Complete problems, which appear to be computationally intractable, is another result in both mathematics and computer science (Cook, 1971; Garey & Johnson, 1979) .

Computer science is like engineering in that it is often concerned with questions of how to accomplish some technological task rather than with scientific questions of why some natural phenomenon occurs. Like mathematics, computer science is largely concerned with manipulating abstract symbols. In contrast to natural science, stories of serendipitous discovery are uncommon: computers often do unexpected things, but rarely in a way that leads to new discoveries. In a company, problems can come from commercial motivations such as the desire to enter into a new market, to improve on an existing product, or to overcome some difficulties that are preventing a project from being delivered. In the academic world, questions often arise from reading already published papers that mention extant problems.

There is in addition a source of problems that is especially important in computer science. Computers are unusual machines in being not only tools but tools for making tools. What computer scientists study is also a machine that can help with its own investigation, as if microbiologists studied their electron microscopes rather than bacteria. In a typical scenario, a computer scientist becomes frustrated with a repetitive, boring, and difficult task that might be relieved by new technology. When someone with the right background and ideas experiences such frustration, creative contributions to computer science can be made.

A major example occurred during the final months of work on the ENIAC, the first general purpose electronic computer, started at the Moore School of Electrical Engineering in 1943 (Goldstine, 1972). ENIAC had been built with the assumption that only one problem would be run on it for a long period of time, so it was difficult and time-consuming to reprogram; in essence it had to be rewired by physically plugging in wires to make a new circuit. This led to great frustration, especially when the team's machine was pitted against the much slower and more primitive Harvard-IBM Mark I machine, which did not use vacuum tubes: "To evaluate seven terms of a power series took 15 minutes on the Harvard device of which 3 minutes was set-up time, whereas it will take at least 15 minutes to set up ENIAC and about 1 second to do the computing" (Goldstine, 1972, p. 198-199).

The solution to this problem, primarily due to John von Neuman, was the concept of a stored program, which enabled the instructions for the computer to perform in the same memory as the data on which the instructions were to operate. With stored programs, a computer could be set to work on a new problem simply by feeding it a new set of instructions. The key insight was that instructions could be treated just the same as data in an early memory device in which numbers were stored as sound waves echoing back and forth in tubes of mercury.

Another example of frustration-based creativity is the invention of the first high-level language, FORTRAN, by John Backus in 1953, which allowed computer programs to be written in comprehensible, algebra-like commands instead of assembly code (Shasha, 1995, pp. 5-20). Assembly code consists of lists of thousands of inscrutable three-letter commands, designed to communicate with the machine at its lowest level. Backus was a programmer at IBM, who said later: "Assembly language was time-consuming; it was an arduous task to write commands that were specific to a particular machine and then to have to debug the multitude of errors that resulted" (Shasha, 1995, p. 10). The layer of abstraction that FORTRAN placed between the underlying electronics and the human was very important to the subsequent development of computer science. Programs could be written faster and with fewer errors, making possible larger and more complex uses of the computer, including building yet more powerful tools and programming languages.

There are many other examples of creativity inspired by frustration in computer science, such as the Unix operating system, Larry Wall's programming language Perl, and Donald Knuth's typesetting system T_EX. Larry Wall declared that the three virtues of the computer programmer are "laziness, impatience, and hubris" (Wall, 1996, p. xiii). It is thus clear that frustration with the limitations of a machine or the repetitive nature of a task is an important motivating force in producing creative answers in computer science. For a survey of important innovations in computer software, see Wheeler (2001).

Although technological frustration may be the origin of many problems in computer science, we should not neglect the intrinsic pleasure for many people of building computers and writing computer programs. According to Brooks (1982, p. 7): "The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures." The term "hacker" referred originally not to people who use programs for destructive purposes such as breaking into the computers of others, but to creative young programmers who reveled in producing clever and useful programs (Levy, 1984; Raymond, 1991). Presumably there are also computer scientists whose creativity is fueled by financial interests: excitement can derive from the prospect that novel software and hardware can make the inventor rich.

Not all computer scientists are concerned with practical problems. The theory of NP-completeness originated because of Stephen Cook's interest in mathematical logic, although he was also interested in computational implications (Cook, personal communication, July 4, 2002): "My advisor at Harvard, Hao Wang, was doing research on the decision problem for predicate calculus. Validity in the predicate calculus is r.e. complete, and this was what gave me the idea that satisfiability in the propositional calculus might be complete in some sense." Thus Cook formulated a new problem in theoretical computer science by analogy to a familiar problem in mathematical logic. Analogies can also be a fertile source of solutions to problems.

3. Creative Analogies in Computer Science

Once computer scientists have posed a problem, how do they solve it? Creative problem solving about computers requires all the cognitive processes that go into scientific research, including means-ends reasoning with rules, hypothesis formation, and generation of new concepts. This section will focus on one important source of creative problem solving, the use of analogies. Analogy is far from being the only source of creative solutions in computer science, but its importance is illustrated by many historical examples.

Following Dunbar (2001) and Weisberg (1993), we distinguish between *local* analogies, which relate problems from the same domain or from very similar domains, and *distant* analogies, which relate problems from different domains. Distant leaps from one domain to another have contributed to important scientific discoveries such as Benjamin Franklin's realization that lightning is a form of electricity (Holyoak & Thagard, 1995, ch. 8). Distant analogies have also played an important role in the history of computer science, but local analogies have also been important for creativity in computer science.

An early distant analogy in computer science was made by Charles Babbage when he realized that the punch cards that controlled the intricate weaving on the automatic Jacquard looms of the day might be used with his hypothetical computing machine, the Analytical Engine (Goldstine, 1972, ch. 2). Lady Ada Lovelace, his friend and collaborator, wrote, "We may say most aptly that the Analytical Engine weaves algebraic patterns just as the Jacquard-loom weaves flowers and leaves" (Goldstine, 1972, p. 22).

Biological analogies have been particularly important in computer science: Von Neumann in 1945 was influenced by McCulloch and Pitts' theories of how the human nervous system could be described mathematically, and used their notation in his "First Draft of a Report on the EDVAC" (Goldstine, 1972, p. 196-197). Another example is the invention by Bob Baran of the packet-switching network, the technology underlying the Internet (Hafner & Lyon, 1996). In the context of the nuclear fears of the time, he wanted to build a network that could survive with numerous nodes knocked out. He took for his inspiration the structure of the human brain with its redundancy and decentralization. The massively parallel Connection Machine of Danny Hillis was built to emulate the parallelism in the brain (Shasha, 1995, p. 188-206). More recently, genetic algorithms and neural networks have proved to be powerful biologically-based ideas that are gradually being incorporated into the mainstream of computer science.

The Wright brothers, in building the first manned, powered flying machine, also used a biological analogy, modeling the turning system of their aircraft after that used by birds (Weisberg, 1995, p. 132-148). However, the Wright brothers also drew on decades of work on flight, carefully studying analogs close to their goal such as sophisticated gliders and attempts at powered flights that had preceded their attempt. The same phenomenon is also found in computer science, where original ideas are often variations and improvements on existing technologies that furnish local analogies.

Eckert and Mauchly had a local inspiration, in the form of John Atanasoff's ABC computer (Slater, 1987, p. 59-63). The ABC was a special-purpose machine, designed only to solve differential equations, and there is some argument about whether it was fully operational, but it was the first of its kind, using vacuum tubes to perform its operations. Mauchly was keenly interested in the machine, and came to visit Atanasoff for four days, talking computers with him the whole time. Only after the mountain of publicity on the ENIAC's unveiling in 1946 did Atanasoff realize how many ideas they had borrowed from his machine. In 1975, at the end of a drawn out lawsuit sparked by the pair's attempts to defend patents on their invention, Atanasoff was declared the legal inventor of the automatic electronic digital computer.

Whether or not this was a case of intellectual plagiarism, it is typical of how inventors absorb technological ideas and build upon them. FORTRAN was preceded by Speedcoding and A-0, and Babbage was familiar with the machines of Leibniz and Pascal. Creative inventors do not work in a vacuum, but are usually very aware of other creative work going on in their field. Atanasoff said later: "What each man accomplishes depends on his brains and energy, but also on the surroundings in which he works…In a larger sense, no man invents anything; he builds and extends a little with his friends and on the shoulders of others" (Huisman, 1999).

Similarly, in 1979 Steven Jobs and engineers from Apple Computers took a tour of the Xerox Palo Alto Research Center, where they saw menus, windows, and user-friendly word processors years before they would be available to the public. At least one member of the team has regretted the effect of the story on the perception of the creativity of the Macintosh group, saying, “Those one and a half hours tainted everything we did, and so much of what we did was original research” (Hiltzik, 1999, p. 343n). Later Microsoft borrowed many of the same ideas for its Windows operating system, another case of problem solving with local analogies.

What then is the role of distant analogies if most important inventions have local antecedents? Consider the case of the invention of object-oriented programming by Alan Kay in the early 70s. As a graduate student, he was exposed to a pioneering computer graphics program called Sketchpad, which had the concept of "master" drawings and "instance" drawings, and the computer language Simula. He was struck with the ideas in Simula, and it stimulated his thinking along analogical lines: “When I saw Simula, it immediately made me think of tissues...[it] transformed me from thinking of computers as mechanisms to thinking of them as things that you could make biological-like organisms in” (Kay, 1996b, p. 575). Kay gave this description of how the cumulative effect of the various examples he was exposed to led to the ideas that culminated in Smalltalk: “The sequence moves from 'barely seeing' a pattern several times, then noting it but not perceiving the 'cosmic' significance, then using it operationally in several areas; then comes a 'grand rotation' in which the pattern becomes the center of a new way of thinking” (Kay, 1996a, p. 514). Many of the parts of Smalltalk can be traced back to local analogies with Simula and Sketchpad, but the general principles of object-oriented programming that were crucial to the final design of Smalltalk did not occur to Kay until he had seen the distant biological analogy. “The big flash was to see this as biological cells. I'm not sure where that flash came from but it didn't happen when I looked at

Sketchpad. Simula didn't send messages either" (Shasha, 1995, p. 43). Similarly, when James Gosling invented the programming language Java, he was inspired both by local analogies, particularly to the programming language C++, and also by a distant analogy to networks of sound and light he experienced at a rock concert (Thagard and Croft, 1999).

Distant analogies in computer science are useful not only for generating new ideas, but also in communicating ideas and suggesting deeper principles underlying the accumulation of technological ideas. Notice the reliance on metaphorical names for the new entities, such as "files" and "folders", "words" and "pages" of memory, and even stranger appropriations, such as "firewall" and "zombie". Some such as "software engineering" and "virus" began as metaphors but later expanded and conceptually deepened the source words, so that there are now more kinds of engineering and more kinds of viruses than there used to be.

4. Everyday Creativity

The previous sections have described some of the most important historical cases of creativity in computer science, but have ignored the day-to-day creative processes of working computer scientists in both academia and industry. For insight into this more prosaic topic, we have examined surveys compiled by *Crossroads*, the on-line student magazine of the Association for Computing Machinery (2002). This magazine includes a feature called "A Day in the Life of…" that interviews practicing computer scientists. We have looked at their answers to two questions: "What do you do to get yourself thinking creatively?" and "What is your problem solving strategy?" The answers were often brief, and sometimes seemed to reflect different interpretations of the questions, but display some interesting patterns. Out of the people profiled in the “A Day in the Life of…” feature, we chose 50 with either an academic position in computer science or a job title of "computer scientist" in industry. All quotations without references are taken from this source.

We have not found evidence that the creativity behind famous inventions is fundamentally different from the everyday creativity of ordinary computer scientists. Not all the respondents to the *Crossroads* questions are ordinary, for they include Herbert Simon and Internet pioneer Leonard Kleinrock. One famous name in computer science, Leslie Lamport, said: “When I look back on my work, most of it seems like dumb luck-- I happened to be looking at the right problem, at the right time, having the right background" (Shasha, 1995, p. 138). This quote underemphasizes the important roles of motivation and intelligence in achievement, but plausibly suggests that creativity is not a rare or supernatural gift.

Only two of the fifty examined respondents to the creativity survey mentioned analogies, and those went no further than simply “think of analogies”. The others either did not use analogies in their creative work or were not aware of using them.

In the answers to the survey’s creativity questions, two modes of creative work were evident, usually one or the other being emphasized in a particular person's answer. We may call them the *intense* mode and the *casual* mode, and they are illustrated by two images of the artist or scientist at work. In the intense mode, the creative individual is hunched over a desk, scribbling madly on pads of paper. In the casual mode, the researcher is lying in the bath relaxing when a solution to a laid-aside problem suddenly hits.

The intense mode is the one that most looks like work, and it is characterized by either writing on many sheets of paper or else animated conversation. This writing is very different from the work involved in communicating one's ideas. The result is not a draft of a paper, but pages covered in scrawls and doodles, showing the evidence of prolonged attacks on an idea from many directions at once: experimentation, examples, pictures, lists, restatements of the problem, anything that could help crack it open. It is most often seen as problem solving, and work in this mode feels relatively rational and systematic. Many of the respondents used systematic language like "first lay out all the options" "ask myself questions" and "create a list of the various issues", and two even provided a numbered list of the steps they take in solving a problem, such as "2. Determine the components and parameters of the problem."

For thinkers in intense mode, the pencil and paper provide a kind of feedback loop, an extension of normal human capabilities that helps to capture thoughts and preserve them beyond the span of short term memory. Other people can also serve to record and amplify ideas. The phrases "bouncing ideas off" and "brainstorming with" were often used in the computer scientists' thoughts about their creative method. Social mechanisms to encourage such creative communication are found in both academic and corporate workplaces.

FORTRAN inventor John Backus once said: "It's wonderful to be creative in science because you can do it without clashing with people and suffering the pain of relationships" (Shasha, 1995, p. 20). But computer scientists depend on social relationships to foster their creativity as much as other researchers. Twelve of the respondents to the survey explicitly mentioned the importance of talking to people in their creative process, and in the last five years of the *Journal of the Association for Computing Machinery* (January 1997 to April 2002) 132 out of 164 articles (80.5%) had more than one author.

It is surprising that, for this group of experienced computer scientists, the tool of choice for focused assault on a problem is usually ordinary writing materials such as pencil and paper. "I probably do best by writing stuff down, often free-flow." "Lots of writing on paper." "Open my notebook to a blank sheet of paper." "[Problem-solving] usually takes A LOT of paper to scribble on, some is just doodling while you stare at the pieces, others is actual notes that help with solutions." Others mentioned dry-erase whiteboards, ubiquitous in every high tech company and research institute, as critical to their creative process, sometimes even using them as a verb, as in "Whiteboard it!" Typical is a description of Xerox PARC whiteboards covered “with boxes filled with other boxes and arrows pointing to yet more boxes with pointers across to new boxes and so on” (Hiltzik, 1999, p. 225).

Why are computers considered inadequate for the most critical part of the development of creative ideas? Part of the reason may be the distracting clutter of computer displays. Another reason may be the comparable robustness and portability of a notepad and pen. But Robert Smith's answer to this question suggests there may be deeper problems: “It is very difficult to use a computer when you are trying to birth new ideas. The computer limits your ideas to the "shape" of documents you know how to create (immediately turns everything into a text document or briefing slide). Blank paper lets you develop an idea in any form you can scribble.” Perhaps someday there will be computer software that incorporates such superior characteristics of plain paper as the lack of distracting features, the ease of movement between text, formulas, and drawings, and the freedom to lay out the page flexibly.

5. The Casual Mode of Creativity

In contrast to the intense mode, the casual mode of creative thinking usually involves inspiration striking during a break from work. Many of the *Crossroads* respondents reported that their best insights occurred not when they were diligently working at their desks, but rather in non-work situations. Many mentioned their time running or hiking as prime creative time, and nine believed physical exercise was important for creative thinking. One reports this strategy for creative thinking: "Go to the beach and run and run and run! Then think and jot things down quickly, saving the refinements for later." The father of computer science, Alan Turing, was an avid long distance runner, often running the 50 km between two universities. According to his biographer, Andrew Hodges, he came to his idea of the Turing Machine lying in the middle of a field after a long run. (Hodges, 1983, p. 100). Other respondents mentioned hiking, bike riding or martial arts as important to helping them get thinking creatively.

Driving was emphasized by one respondent: “I drive an hour each way to work which gives me a lot of time to contemplate and just let my mind range from topic to topic. Most of the time I spend the drive looking at the scenery and then seeing where that topic takes me.” Other researchers also report that their daily drive or subway ride often constitutes their most important creative period of the day.

The shower also seems to be an excellent place for generating creative ideas. One reported: "Once I have a problem, it becomes part of me, and ideas come up mostly in non-work situations. I came up with a major idea for my thesis while in the shower." Alan Kay, the inventor of object-oriented programming and the graphical user interface, believed he got his best ideas in the shower. He even had one to himself in the basement of the Xerox PARC research facility where he made some of his most startling inventions, using it in the morning, his most productive time of the day (Hiltzik, 1999, p. 217).

Finally, several computer scientists report getting great ideas in bed, in the middle of the night or on waking. One reported her response to getting badly stuck on a problem is to go to bed and "wake up at 2 AM with the solution suddenly obvious." The original inventor of public key cryptography, J.H. Ellis, reportedly made his discovery in the middle of the night, after spending some days mulling over the problem of how to allow secure communication by two parties who had never met (Ellis, 1987).

These occasions of casual mode creativity share the following characteristics: immersion in a problem domain, absence of immediate pressure to focus on the problem, absence of distractions, mental relaxation, unstructured time, and solitude. Physical activities are an important way of reaching a relaxed state of mind where ideas that had been studied intensely before are free to combine in various ways without immediate pressure to solve a problem. This increased freedom helps to overcome problems that have become obstacles.

An important illustration of the casual mode in computer science history is the story of John Atanasoff (Slater, 1987, p. 53-63). Atanasoff was a professor at Iowa State College and had become very concerned with the problems his graduate students were having doing calculations with the primitive analogue machines available at the time. He had thought intensely about the problems of computing, but felt stuck. He went for a very fast drive in his car until he crossed the border into Illinois, where he stopped at a roadhouse and ordered a drink. “Now, I don't know why my mind worked then when it had not worked previously, but things seemed to be good and cool and quiet.... I would suspect that I drank two drinks perhaps, and then I realized that thoughts were coming good and I had some positive results” (Mackintosh, 1988). In the few hours he spent at the roadhouse, he formed several important ideas for his computer, including the use of binary numbers and the use of vacuum tubes instead of relays to increase the speed of calculations. Atanasoff described the feeling of having heightened creativity, of being inspired: “I remember it perfectly, as if it were yesterday. Everything came together in that one night. All of a sudden I realized that I had a power that I hadn't had before. How I felt that so strongly in my soul, I don't know. I had a power--I mean I could do things, I could move, and move with assurance” (Slater, 1987, p. 55).

The casual mode contributes to the phenomenon of sudden insight, described by Hadamard (1945, p. 21) as “those sudden enlightenments which can be called inspirations”. He quoted Helmholtz: "Creative ideas...come mostly of a sudden, frequently after great mental exertion, in a state of mental fatigue combined with physical relaxation." Hadamard proposed that creative insights happen in four stages, which he called preparation, incubation, illumination, and "precising". But the incubation stage, where little directed thought is given to the problem, could only lead to illumination when preceded by hard work on the problem. As Louis Pasteur said, chance favors the prepared mind. Thus insight in the casual mode requires previous work in the intense mode.

According to Jack Goldman, who was head of research at Xerox and responsible for the famously creative computer science lab at the Palo Alto Research Center: “Invention can result from a flash of genius or painstaking pursuit of a technical response to an identified or perceived need--sometimes perceived only by the inventor himself” (Alexander & Smith, 1988, p. 34). Although the intense mode feels less special and creative than when ideas come out of the blue in the casual mode, it is a critical part of creative work, and responsible for many important discoveries. Creative researchers typically construct their days to give some time to both modes.

In the *Crossroads* survey, some computer scientists report getting stimulated creatively by any kind of creative work, not just work in computer science and its related disciplines. “A good film or art exhibit or any creative work done with excellence can inspire me as much as computing.” “Reading someone else's creative material starts the juices flowing again.” “Reading articles in Science magazine about biology or something else completely outside of what I usually do is also stimulating.” This seems to be more than just a case of noticing distant analogies between whatever one is reading and one’s own work. Why should creative work in one domain inspire creativity in a researcher in a totally different domain? Perhaps inspiration derives from stimulation of excitement that facilitates work in general; Isen (1993) reviews research that shows that induction of positive mood enhances creative decision making.

6. Comparison with Natural Science

We now compare what we have learned about creativity in computer science with aspects of creativity in the natural sciences. Problems in physics, chemistry, and biology can take the form of several different kinds of questions. Some questions are empirical, inquiring about the relations between two or more variables with observable values. On the more theoretical side, there are why-questions that ask for an explanation of observed empirical relations. Sciences such as biology often also ask how-questions that can be answered by specifying a mechanism that shows how observed functions are performed. Finally, applied science such as biomedicine asks how-questions concerning how what is known about mechanisms might be used to solve some practical problem such as treating disease. Computer science is not concerned with empirical questions involving naturally observed phenomena, nor with theoretical why-questions aimed at providing explanations of such phenomena. Nor does it study naturally occurring mechanisms, but rather usually aims at producing new mechanisms, both hardware and software, that can provide solutions to practical problems. This is computer science as engineering. The problems in theoretical computer science are more like problems in pure and applied mathematics than like problems in the natural sciences. They usually involve very abstract notions of computation and can often be only tangentially relevant to practical concerns of computing.

In the natural sciences, the origins of problems are often empirical, when surprising observations lead to experimentation and theorizing aimed at replacing surprise by understanding. Some problems arise for conceptual reasons, for example when two plausible theories are incompatible with each other. The motivation for theoretical computer science is largely conceptual, but much work in computer science originates with practical frustrations with available technology. Thus the origins of problems in most computer science are like the origins of problems in applied sciences such as biomedicine. Many emotions, including frustration, need, surprise, interest and ambition can motivate novel work in both science and engineering (Thagard, 2002). We said above that there seems to be less serendipity in computer science than in natural science, so perhaps frustration is a more important emotion than surprise for technological innovation.

How does the finding of solutions to problems in computer science compare with finding solutions to the empirical and theoretical problems in the natural sciences? Thagard and Croft (1999, p. 134) compared scientific discovery with technological innovation, and concluded:

> Scientists and inventors ask different kinds of questions. Because scientists are largely concerned with identifying and explaining phenomena, they generate questions such as:
>
> Why did X happen? What is Y? How could W cause Z?
>
> In contrast, inventors have more practical goals that lead them to ask questions of the form:
>
> How can X be accomplished? What can Y be used to do? How can W be used to do Z?
>
> Despite the differences in the form of the questions asked by scientists from the form of the questions asked by inventors, there is no reason to believe that the cognitive processes underlying questioning in the two contexts are fundamentally different. Scientists encounter a puzzling X and try to explain it; inventors identify a desirable X and try to produce it.

In computer science the desirable X is a better way of computing. Like all problem solvers, computer scientists use means-ends reasoning and analogies to generate solutions to problems. Hence creativity in computer science seems very similar cognitively to creativity in thinking in the natural sciences. For further discussion of the cognitive processes involved in scientific creativity, see Thagard (1988, 1992, 1999).

We are not aware of any systematic studies of the roles of intense and casual modes of thinking the natural sciences. Cognitive psychologists have primarily investigated the intense mode, but there is anecdotal evidence that the casual mode contributes to creativity in the natural sciences. For example, Kekulé reported apprehending the structure of benzene during a dream. It would be interesting to conduct a survey, analogous to the *Crossroads* interviews with computer scientists, to determine whether other kinds of thinkers are also sometimes inspired in casual contexts. It would not be surprising if natural scientists also can benefit from physical exercise, showers, concerts, and lying in bed. (Thagard got one of his best ideas, the computational theory of explanatory coherence, while watching a boring movie, *Beverly Hills Cop 2*.)

In sum, the cognitive and social processes that foster creativity in computer science do not seem to be different in kind from the processes that underlie success in the natural sciences, especially applied sciences such as medicine. Computer science is certainly different in that it studies an artifact, computers and computation, rather in than naturally occurring phenomena. But creativity science, technology, and mathematics all involve asking novel questions to pose hard problems and then answering them by using analogies and other kinds of reasoning to form new hypotheses and concepts.

7. Conclusion

We conclude by summarizing the major findings of our investigation of creativity in computer science. Problems in computer science originate in both engineering and mathematical contexts. Many engineering problems in computer science arise from frustration with available computational tools, but creativity can also be fueled by financial interests and the pleasure of building computers and writing software. Both local and distant analogies can be useful in solving problems. Everyday creativity in computer science operates in both an intense mode of focused concentration and a casual mode in which inspiration strikes during non-work activities. Answering computational questions seems to involve the same kinds of cognitive processes that generate answers to empirical questions in the natural sciences. Creativity requires caring about a problem sufficiently to work on it intensely, using analogies and other means of generating novel solutions.

Acknowledgements. For helpful suggestions, we are grateful to Peter Buhr, Gordon Cormack, Amyrose Gill, Patrick Gill, Ming Li, and C. I. Stolarski. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada.

References

Alexander, R.C. & Smith, D.K. (1988) Fumbling the future : How Xerox invented, then ignored, the first personal computer. New York: W. Morrow.

Association for Computing Machinery (2002). A day in the life of.... Retrieved April 15, 2002, from http://www.acm.org/crossroads/dayinlife/index.html

Brooks, F. P. (1982). The mythical man-month: Essays on software engineering. Reading, Mass.: Addison-Wesley.

Cook, S. A. (1971). The complexity of theorem proving procedures. In ACM Special Interest Group on Algorithms and Computation Theory, Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158). New York: ACM Press.

Dunbar, K. (2001). The analogical paradox: Why analogy is so easy in naturalistic settings, yet so difficult in the laboratory. In D. Gentner, K. J. Holyoak, & B. K. Kokinov (Eds.), The analogical mind (pp. 313-334). Cambridge, MA: MIT Press.

Ellis, J. H. (1987). The history of non-secret encryption. Retrieved March 20, 2002, from http://www.cesg.gov.uk/publications/media/nsecret/ellis.pdf

Frenkel, K. A. (1987). Profiles in computing: Brian K. Reid: a graphics tale of a hacker tracker. Communications of the ACM, 30, 820-823.

Garder, M.(1978). Aha! Insight. New York: Scientific American/W.H. Freeman.

Garey, M., & Johnson, D. (1979). Computers and intractability. New York: Freeman.

Goldstine, H. (1972). The computer from Pascal to Von Neumann. Princeton, N.J.: Princeton University Press.

Hadamard, J. (1945). The psychology of invention in the mathematical field. New York: Dover Publications.

Hafner, Katie, & Lyon, Matthew (1996). Where wizards stay up late: The origins of the internet. New York: Simon & Schuster.

Hiltzik, M. A. (1999). Dealers of lightning: Xerox PARC and the dawn of the computer age. New York: HarperCollins.

Hodges, A. (1983). Alan turing: The enigma. London: Burnett Books.

Holyoak, K. J., & Thagard, P. (1995). Mental leaps: Analogy in creative thought. Cambridge, MA: MIT Press/Bradford Books

Hughes, T. P. (1983). American genesis: A century of invention and technological enthusiasm.

Huisman, D. (1999, Oct/Nov). Stories behind John Vincent Atanasoff's computer. The Academic Information Technologies newsletter 33 (65). Retrieved April 15, 2002, from http://www.ait.iastate.edu/newsletter/199910/article4.html

Isen, A. M. (1993). Positive affect and decision making. In M. Lewis & J. M. Haviland (Eds.), Handbook of emotions (pp. 261-277). New York: Guilford Press.

Kay, A. C. (1996a). The Early History of SmallTalk. In Bergin, T. J., & Gibson, R.G. (Eds.), History of programming languages II (pp. 511-579). New York: Addison-Wesley.

Kay, A. C. (1996b). Transcript of SmallTalk Presentation. In Bergin, T. J., & Gibson, R.G. (Eds.), History of programming languages II (pp. 579-589). New York: Addison-Wesley.

Knuth, D. E. (1974). Computer programming as an art. ACM turing award lectures: The first twenty years, 1966 to 1985. New York: ACM Press.

Levy, S. (1984). Hackers: Heroes of the computer revolution. Garden City, N.Y.: Anchor Press/Doubleday.

Mackintosh, A. R. (1988). Dr. Atanasoff's computer. Retrieved April 15, 2002, from http://www.geocities.com/computerresearchassociated/Atanasoff.htm.

Raymond, E. S. (1991). The new hacker's dictionary. Cambridge, Mass.: MIT Press.

Shasha, D. (1995). Out of their minds: The lives and discoveries of 15 great computer scientists. New York: Copernicus.

Slater, R. (1987). Portraits in silicon. Cambridge, Mass.: MIT Press.

Thagard, P. (1988). Computational philosophy of science. Cambridge, MA: MIT Press/Bradford Books.

Thagard, P. (1992). Conceptual revolutions. Princeton: Princeton University Press.

Thagard, P. (1999). How scientists explain disease. Princeton: Princeton University Press.

Thagard, P. (2002). The passionate scientist: Emotion in scientific cognition. In P. Carruthers & S. Stich & M. Siegal (Eds.), The cognitive basis of science (pp. 235-250). Cambridge: Cambridge University Press.

Thagard, P., & Croft, D. (1999). Scientific discovery and technological innovation: Ulcers, dinosaur extinction, and the programming langage Java. In L. Magnani, N. Nersessian & P. Thagard (Eds.), Model-based reasoning in scientific discovery (pp. 125-137). New York: Plenum.

Wall, L. (1996). Programming Perl (2nd edn). Sebastopol, California: O'Reilly.

Weisberg, R. W. (1993). Creativity: Beyond the myth of genius. New York: W.H. Freeman.

Wheeler, D. A. (2001). The most important software innovations. Retrieved April 15, 2002, from http://www.dwheeler.com/innovation/innovation.html