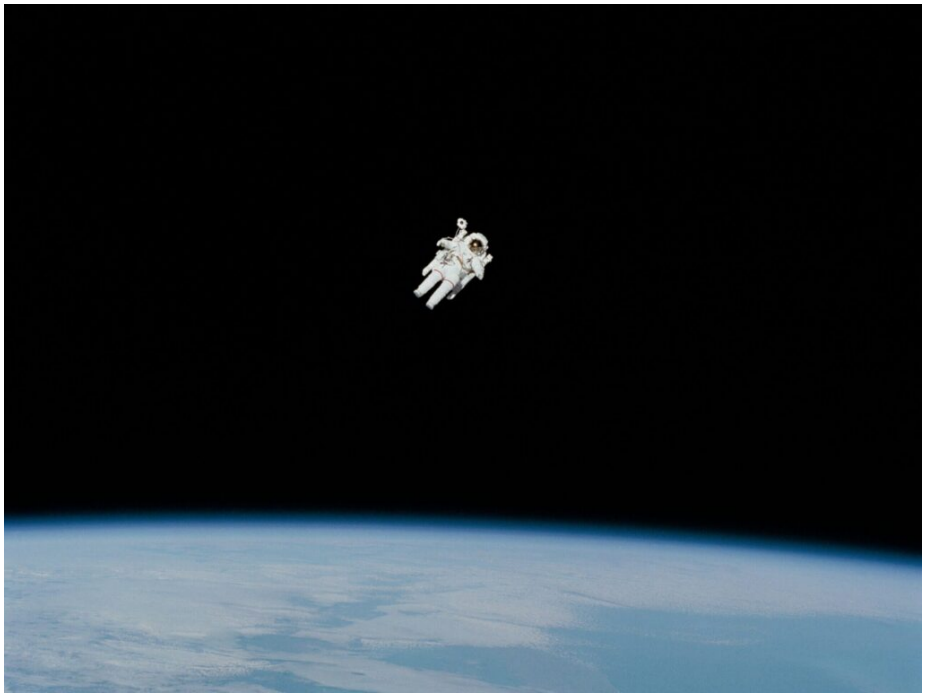
Issue 063: Space

Adrian Kosmaczewski

December 4th, 2023



Welcome to the sixty-third issue of *De Programmatica Ipsum*, about *Space.*

In this edition:

- We ask ourselves how much rocket science[1] is required to write quality code.
- In the Library section[2], we review the impact of "Cosmos" by Carl Sagan[3] in our society.
- In our Vidéothèque section[4], we listen to Margaret Hamilton[5] telling us stories about software engineering and Moon landings.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, and Szymon Licau.

Enjoy this issue! Please subscribe to our free newsletter[6] to stay updated about new releases,

[1] https://deprogrammaticaipsum.com/rocket-science/
[2] https://deprogrammaticaipsum.com/category/library/
[3] https://deprogrammaticaipsum.com/carl-sagan/
[4] https://deprogrammaticaipsum.com/category/videotheque/
[5] https://deprogrammaticaipsum.com/margaret-hamilton/
[6] https://deprogrammaticaipsum.com/newsletter/

share the articles on social media, or contribute[7] if you would like to support our work with a donation via Liberapay[8].

Cover photo by NASA[9] on Unsplash[10].

[7] https://deprogrammaticaipsum.com/contribute/

[8] https://liberapay.com/

[9] https://unsplash.com/@nasa?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[10] https://unsplash.com/photos/astronaut-in-spacesuit-floating-in-space-Yj1M5riCKk4?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Rocket Science

Adrian Kosmaczewski

December 4th, 2023

Space exploration and computers are inextricable. It is unthinkable, at least with our level of understanding, to imagine one without the other. It is no surprise that both grew together, mutually reinforcing one another, precisely at the same point in human history. Being the most complex undertaking ever attempted by Humanity, involving the juggling of so many balls in the air at once, computers became our allies to make space travel a reality.

Leaving aside the obvious risk of shipping humans inside tin cans traveling at supersonic speeds, the lives of astronauts and cosmonauts depend on solid software to understand what is going on. Or, at the very least, how far they are from Earth and how to come back. At the risk of stating the obvious, let us remind ourselves that this is still the only place we can inhabit in the known Universe, even if we are doing everything possible to render it as unlivable as possible.

Arguably, one of the most brilliant creations of George Lucas in his *Star Wars* universe is the idea of "Astromech droids"[1]; small robots perfectly sized to fit *ad hoc* slots on spacecraft, able to communicate with the onboard computers, and acting as a required interface for the organic pilots at the helm starring in the films, human or not. So good are Astromech droids at their craft, that even C-3PO[2], supposedly "fluent in over six million forms of communication," has trouble understanding the "strange dialect" used by the Millenium Falcon[3] onboard computer during the most tense moments of *The Empire Strikes Back*. It raises the question of the effectiveness of the standards body in charge of that galaxy far, far away; are there no mandatory reviews to follow before takeoff? Ah right, the Falcon is a smuggling ship. Move on, there is nothing to see here.

Onboard computers that are difficult to communicate with are a staple in science fiction.

[1] https://starwars.fandom.com/wiki/Astromech_droid
[2] https://en.wikipedia.org/wiki/C-3PO
[3] https://starwars.fandom.com/wiki/YT-1300_light_freighter

Comes to mind the quintessential onboard computer, HAL 9000[4], who (who or which?) essentially interacted with humans in plain English[5]. Such fluency does not prevent it from becoming a pain in the neck for Dave Bowman, but that is another story. Another classic example is the malfunctioning computers (and their hilarious interaction with crew members) in charge of "Thermostellar Triggering Devices" carried by the Dark Star ship in the underrated and excellent John Carpenter's 1974 film of the same name[6]. The M-5 computer[7] featured in the 24th chapter of the second season of the original Star Trek series (1968) deserves another special mention.

How good the software used in space travel needs to be? The (almost obvious) answer shows up in a classic, memorable, and certainly recommendable 1996 article[8] from Fast Company magazine (archive[9]) whose title references an equally epic 1983 motion picture[10]:

> The right stuff is the software. The software gives the orders to gimbal the main engines, executing the dramatic belly roll the shuttle does soon after it clears the tower. The software throttles the engines to make sure the craft doesn't accelerate too fast. It keeps track of where the shuttle is, orders the solid rocket boosters to fall away, makes minor course corrections, and after about 10 minutes, directs the shuttle into orbit more than 100 miles up. When the software is satisfied with the shuttle's position in space, it orders the main engines to shut down — weightlessness begins and everything starts to float.
>
> But how much work the software does is not what makes it remarkable. What makes it remarkable is how well the software works. This software never crashes. It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats : the last three versions of the program — each 420,000 lines long-had just one error each. The last 11 versions of this software had a total of 17 errors. Commercial programs of equivalent complexity would have 5,000 errors.

We are in 2023, which means that the article above is 27 years old. Is it fair to compare your containerized Cloud Native or Android application to the software piloting the latest spaceship? (The Space Shuttle ended its career meanwhile, sadly.) One needs to make a fair comparison. A few months before the publication of this article in Fast Company, the Ariane 5 blew up[11] because of a badly handled integer overflow exception. And three years later[12],

> NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

OK, OK, you might argue that the last one is not *strictly speaking* a software error; the actual issue was related to communication and quality management. Yes, but those are the hardest parts of software engineering. Any programmer can agree with this author that the simplest part of a project is actually pumping out the code on the IDE.

[4] https://en.wikipedia.org/wiki/HAL_9000
[5] https://deprogrammaticaipsum.com/the-winner-takes-it-all/
[6] https://en.wikipedia.org/wiki/Dark_Star_(film)
[7] https://en.wikipedia.org/wiki/The_Ultimate_Computer
[8] https://www.fastcompany.com/28121/they-write-right-stuff
[9] https://archive.ph/2021.10.04-011409/https://www.fastcompany.com/28121/they-write-right-stuff
[10] https://en.wikipedia.org/wiki/The_Right_Stuff_(film)
[11] https://en.wikipedia.org/wiki/Ariane_flight_V88
[12] http://edition.cnn.com/TECH/space/9909/30/mars.metric.02/

The aforementioned article on Fast Company contains other gems:

> Software is everything. It also sucks.
>
> "It's like pre-Sumerian civilization," says Brad Cox, who wrote the software for Steve Jobs NeXT computer and is a professor at George Mason University. "The way we build software is in the hunter-gatherer stage."
>
> John Munson, a software engineer and professor of computer science at the University of Idaho, is not quite so generous. "Cave art," he says. "It's primitive. We supposedly teach computer science. There's no science here at all."

The article explains the effectiveness of NASA-made software with four basic premises:

1. Planning. "Our requirements are almost pseudo-code."
2. Teams. Composed of both coders *and* testers, the latter referred to as "verifiers" in the article.
3. Data. The two "enormous databases, encyclopedic in their comprehensiveness" mentioned in the text are just a source control system, and a bug tracker.
4. Culture. "You don't get punished for making errors."

Point 3 above is probably the only one that has been successfully implanted in our minds and industry since 1996. The other three items in the list are a work in progress, at best, or a glaring omission due to budgetary cuts, layoffs, or just the "Agile" word being hypocritically screamed at every meeting (particularly true for those arguments in favor of the first point above).

Page 193 of the July 1985 issue of BYTE Magazine[13] (dedicated to space and astronomy subjects) contains another interesting article about how NASA prepared its missions in the first decade of the Space Shuttle program. In this case, including details about the programming language of choice for the Shuttle Imaging Radar-B (SIR-B)[14] experiment:

> Both the SIR-B mission-planning software and the real-time communications software were written primarily in FORTH. The off-the-shelf FORTH implementation (PC/FORTH by Laboratory Microsystems) included fast display graphics for the IBM PC, a standard PC-DOS file interface, and high-level support for the 8087 coprocessor. Again. speed. adaptability, and readily available support were major considerations in choosing the programming language.

The available choices in the programming languages market 38 years ago seem stunning by today's standards. C++ was still virtually unknown outside Bell Labs, and Perl had not even been released yet.

Fast-forward to 2023, and NASA's Spacecraft Software Engineering Branch[15] keeps on crunching better software than any of us could imagine, and publishing a full catalog[16] of software applications, often open source[17] and ready to download and use in your own spacecraft. Not only that, but as this article hits the web, the 46-year-old Voyager 2 probe, at 20-something *billion* kilometers from Earth, just received a much hyped software

[13] https://archive.org/details/BYTE_Vol_10-07_1985-07_Computers_and_Space/BYTE%20Vol%2010-07%201985-07%20Computers%20and%20Space/page/n207/mode/2up

[14] https://www.jpl.nasa.gov/missions/shuttle-imaging-radar-b-sir-b

[15] https://www.nasa.gov/software-robotics-and-simulation-division/spacecraft-software-engineering-branch/

[16] https://software.nasa.gov/

[17] https://github.com/nasa

patch[18]. Meanwhile, some of us Earthlings have to reboot our computer twice just to print a document. Whatever.

NASA, who uses all kinds of computers[19], has adopted Linux as an operating system of choice, first on the International Space Station[20], and then as support for other missions[21], as a real-time OS[22], on Artemis' flight simulators[23], and even on Mars[24]. There are rumors that the CADRE rovers[25] will run with Debian on our natural satellite. SpaceX took note[26], as we should all.

A quick glance at the frowning eyebrows and shrugs of the readers of this article indicates that no, we have not advanced much in the past 27 years. Paraphrasing Brad Cox[27], we are still in Precambrian times, with the added insult of evolving in an industry that finds periodic reinventions[28] of the wheel not only palatable, but an actual Darwinian *conditio sine qua non* in our evolution towards the future.

Spoiler alert: it is not, and we keep shipping shitty software.

As a society, we could be producing much finer software if we stopped worshiping the wrong gods decade in, decade out. The solution is not Rust, yet another JavaScript framework, Kubernetes, or GitHub Copilot. The solution is not yet another IDE or yet another Agile retrospective (enough of it[29], already). And yes, it is a desirable goal for our civilization to crank out stable, fast, high-quality, maintainable software, not only for *mission-critical* stuff, but even for the most mundane of games. And doing so should not be considered a seemingly impossible task in the order of rocket science, but it could take a few cues from actual rocket scientists.

As programmers, we need to start giving a shit about our software and the way it impacts our modern world. We must stop conflating the creation of software as a simple way to becoming yet another billionaire, but to take our craft with the rigor it deserves, even if that means overcoming our aversion to writing[30] and drafting a spec from time to time.

Or, to put it another way: we do not need hot stuff tonight, Donna[31]; the right stuff will do just fine.

Cover photo by NASA[32] on Unsplash[33].

---

[18] https://arstechnica.com/space/2023/10/nasa-wants-the-voyagers-to-age-gracefully-so-its-time-for-a-software-patch/

[19] https://www.eclipseaviation.com/the-different-types-of-computers-that-nasa-uses/

[20] https://www.extremetech.com/extreme/155392-international-space-station-switches-from-windows-to-linux-for-improved-reliability

[21] https://elinux.org/images/9/94/2017_Leppinen_-_Current_use_of_Linux_in_spacecraft_flight_software.pdf

[22] https://ntrs.nasa.gov/citations/20200002390

[23] https://www.zdnet.com/article/fedoras-in-space-red-hat-helps-with-nasas-artemis-lunar-missions/

[24] https://www.pcmag.com/news/linux-is-now-on-mars-thanks-to-nasas-perseverance-rover

[25] https://www.jpl.nasa.gov/missions/cadre

[26] https://www.zdnet.com/article/from-earth-to-orbit-with-linux-and-spacex/

[27] https://deprogrammaticaipsum.com/brad-cox/

[28] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/

[29] https://deprogrammaticaipsum.com/enough-agile/

[30] https://deprogrammaticaipsum.com/on-the-aversion-to-writing/

[31] https://www.youtube.com/watch?v=KhcaPNuaJNU

[32] https://unsplash.com/@nasa?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[33] https://unsplash.com/photos/space-shuttle-challenger-launches-from-kennedy-space-center-dCgbRAQmTQA?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Margaret Hamilton

Adrian Kosmaczewski

December 4th, 2023

On Sunday, July 20th, 1969, at precisely 20:14:19 UTC[1], just a mere three minutes before touchdown, the voice of Edwin Eugene Aldrin Jr.[2] confirmed the "Go for landing" order received from Mission Control together with a phrase nobody wanted to hear at that moment: "Program alarm – 1201."

A lot has been written about the Apollo Guidance Computer[3], its technical characteristics, and the true nature of alarms 1201 and 1202. On this occasion, we feel the urge to highlight Margaret Hamilton[4]'s monumental contributions to the history of the 20th century: not just the assembly language code that, at the height of the first manned Lunar landing, produced one of the most dramatic error codes ever logged in the history of software engineering, but her groundbreaking idea of transforming mere programming into, precisely, software engineering.

Her assembly code was the one that, printed out, represented a stack of paper as high as Ms Hamilton herself, as shown in that iconic photograph[5] that you most probably have seen somewhere by now. She was responsible for it, as explained by David G. Hoag in a 1976 report titled "The history of Apollo onboard guidance, navigation, and control"[6]:

> Much of the detailed code of these programs was written by a team of specialists led by Margaret Hamilton. The task assignments to these individuals included, in addition to writing the code, the testing to certify that the program element met requirements. Overall testing of the assembled collection of program elements necessarily took the use of considerable human and machine resources.

[1] https://apolloinrealtime.org/11/?t=102:42:19
[2] https://en.wikipedia.org/wiki/Buzz_Aldrin
[3] https://en.wikipedia.org/wiki/Apollo_Guidance_Computer
[4] https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)
[5] https://news.mit.edu/2016/scene-at-mit-margaret-hamilton-apollo-code-0817
[6] https://arc.aiaa.org/doi/epdf/10.2514/3.19795

> The programs had to be as near error-free as possible and any anomalies had to be understood and recorded for possible affect on the mission. Actually, no program errors were ever uncovered during the missions.

And software engineering it was, once again, thanks to Ms Hamilton, who rightfully treated her craft with the respect and seriousness it deserved, thus coining the expression "software engineering" at a time when software did not enjoy a single iota of respect from the computer industry.

Ms Hamilton, after having worked with Edward Lorenz[7] in weather prediction programs, and having ground her programming teeth in aircraft detection algorithms for the SAGE project[8], came up with simple heuristics to craft better code. Among them, adding comments to her code. Shocking. I know, right? She tells the story herself in this 2017 interview[9], conducted by the renowned software journalist David C. Brock[10] (prolific author of articles for the IEEE Spectrum magazine[11], by the way) and published by the Computer History Museum.

> They thought that was so funny that I put comments beside the code. "Why do you put comments beside the code when you can just read the code?" (...)
>
> And that's why I started to do it. Because I thought, "I could forget my own code, because I've been tricky. Now I'm going to trick myself," right.

Or, you know, good old code reviews:

> And somebody else would say, "I used the augekugel method." And I thought, "I never heard of the augekugel method. I got to find out what this is. I can't let them know I don't know the thing that they keep talking about all the time." But I couldn't find out what it was. So finally, I said, "What is this augekugel method that you all talk about using when you solve problems?" Turns out it means eyeballing in German. Scanning, going through the listing, you know, understanding what's going on. (...) I learned if you don't know, ask. Ask questions. Don't be afraid to ask a dumb question. There is not a dumb question. What's important is to learn what you need to learn.

Another useful task in her software engineering belt was to always hope for the best and plan for the worst:

> So I checked it out to see what was happening, and my daughter had selected the pre-launch program when it was in flight (...) And so I came back and told people about it (...) and it was like it kept worrying me and I kept saying, "We've gotta put a fix in there so that if anybody tries to select P01 during flight, if the astronaut makes a mistake, it's gonna say, no, you can't do that. You've selected P01 during flight." (...) And the powers that be (...) didn't want to put that in because they were worried about extra code, and the astronauts would "never, ever make a mistake," quote, unquote. (...)
>
> Well, wouldn't you know, after I was able to get that in (...) they did it. They selected P01 during flight, and that was the Apollo 8.

[7] https://en.wikipedia.org/wiki/Edward_Norton_Lorenz
[8] https://en.wikipedia.org/wiki/Semi-Automatic_Ground_Environment
[9] https://www.youtube.com/watch?v=6bVRytYSTEk
[10] https://computerhistory.org/profile/david-brock/
[11] https://spectrum.ieee.org/u/david-c-brock

Her obsession with handling all possible situations during one of the most critical moments in space exploration had no limits:

> So what I wanted to do was to interrupt the astronauts to tell them there's an emergency (...) the problem was he's got his normal display and now you put up a priority display. Which one is he answering, you see? And so I came up with the idea of counting to five before he answers.

And this is how we reach the 1201 program alarm mentioned earlier:

> So, anyway, now we go to Apollo 11 and it's time to land, okay? And so I'm standing in the SCAMA[12] room again and they're going through all the things you go through for landing, and all of the sudden guess what comes up: 1201 and 1202 priority displays telling them there's an emergency. This is just before they land. And here were the things that I had wanted to do was to warn the astronaut when there's an emergency, and 1201 and 1202 means that there were too many things going on in the computer. One was to do with the tasks, too many tasks trying to get scheduled, and the other was too many jobs based on priority getting scheduled. (...)
>
> And Jack Garman (...) made the decision to go because he knew right away what the 1201 and 1202 alarms were.

Through it all, software engineering was born, in a great example of allyship.

> But we were the programmers, and we were kind of like second-class citizens. (...) I kept noticing patterns between the different kinds of engineering that were very similar. (...) I said, "Why don't we call this one the hardware engineering part and this the software engineering part?" (...) So, anyway, the engineers thought it was funny. "There's Margaret again with her software engineering," you know. (...) and one of the hardware gurus stood up in a big meeting (...) "This is engineering, what you people are doing, just as much as the stuff we're doing," meaning the hardware. He said, "She's trying to formalize it and I think it should become formalized," and everybody respected this hardware guy.

Many thanks to David C. Brock and the Computer History Museum for this interview of Margaret Hamilton. The final result is a precious video, chosen as this month's Vidéothèque choice[13], including a full transcript[14]. Complement this interview with Ms Hamilton's participation[15] at the first conference of the Apollo Guidance Computer History Project in 2001; her letter to the editor[16] of Datamation in 1971 to correct some misconceptions about the nature of the 1201 and 1202 errors; her presentation at ICSE 2018[17]; and her articles "Universal Systems Language: Lessons Learned from Apollo"[18], published in IEEE Computer in December 2008, and "What the Errors Tell Us"[19], featured in the September 2018 edition of IEEE Software.

[12] https://wehackthemoon.com/missions/apollo-11-moon-landing

[13] https://www.youtube.com/watch?v=6bVRytYSTEk

[14] https://www.computerhistory.org/collections/catalog/102738243

[15] https://wayback.archive-it.org/9060/20230418191423/https://authors.library.caltech.edu/5456/1/hrst.mit.edu//hrs/apollo/public/conference1/hamilton-intro.htm

[16] https://web.archive.org/web/20191115232655/https://wehackthemoon.com/people/margaret-hamilton-computer-got-loaded

[17] https://www.youtube.com/watch?v=ZbVOF0Uk5lU

[18] http://www.htius.com/Articles/r12ham.pdf

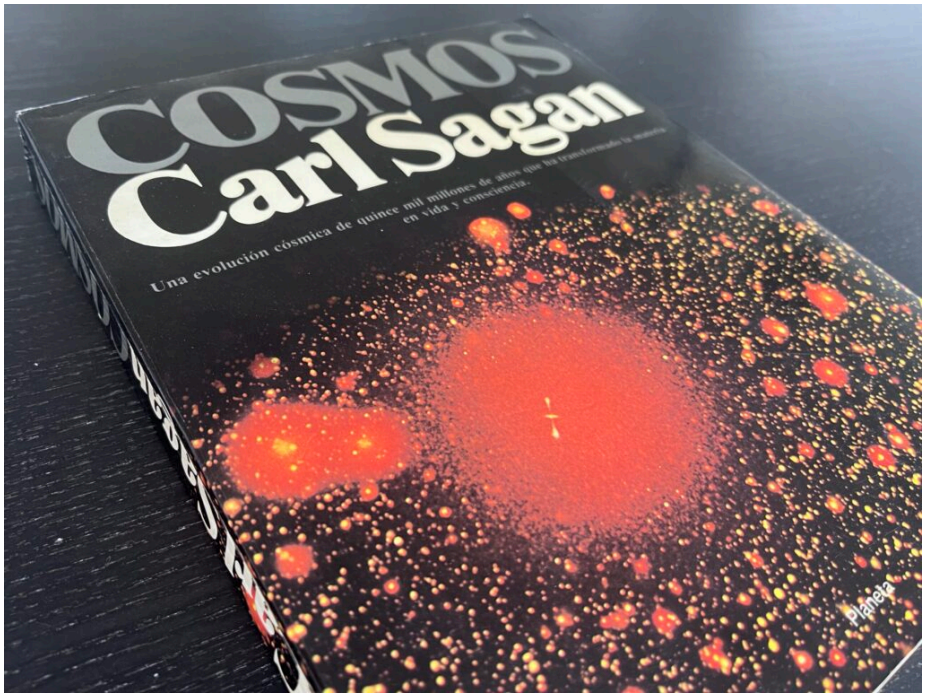
[19] https://ieeexplore.ieee.org/document/8409915

Cover snapshot chosen by the author.

Carl Sagan

Adrian Kosmaczewski

December 4th, 2023



The news of a software patch uploaded[1] to the Voyager probes[2] reminded me of a 1980 book telling precisely the story of how their journey began 46 years ago. When said book hit the publishing press, Voyager 1 had just finished its flyby of Saturn, a planet which Voyager 2 was about to survey a few months later. Assisted by gravity slingshots, the latter probe would reach Uranus in 1986 and Neptune in 1989. Both Voyagers would cross the Heliopause decades later, one in 2012, and the other in 2018. Against all odds, they are both beeping back to Earth as you read these lines.

The book in question is "Cosmos"[3], of course, by the late astronomer Carl Sagan[4]. It is a compendium of history, biology, astronomy, engineering, and social sciences, with breadth and a poetry at each step of the way.

No, there are no extensive tales about computers or programming languages in the pages of "Cosmos", something the readers of this magazine might be expecting every month. But of course, computers there are, controlling the cameras and governing the ship, constantly keeping an eye[5] on some stars for navigation:

[1] https://arstechnica.com/space/2023/10/nasa-wants-the-voyagers-to-age-gracefully-so-its-time-for-a-software-patch/

[2] https://en.wikipedia.org/wiki/Voyager_program

[3] https://en.wikipedia.org/wiki/Cosmos_(Sagan_book)

[4] https://en.wikipedia.org/wiki/Carl_Sagan

[5] https://web.archive.org/web/20151016052108/http://www.au.af.mil/au/awc/awcgate/jplbasic/bsf11-2.htm

> The attitude and articulation control subsystem (AACS) computer manages the tasks involved in stabilization via its interface equipment. For attitude reference, star trackers, star scanners, solar trackers, sun sensors, and planetary limb trackers come into use. Voyager's AACS uses a sun sensor for yaw and pitch reference, and a star tracker trained continuously on a bright star at right angles to sunpoint for roll reference.

Here are some crunchy details[6] computer fans will surely enjoy:

> The Voyager CCS and Viking CCS would ultimately have the same amount of memory (just under 70kB) despite the routines and programs for Voyager being much more complex. In-flight programming allowed for new routines and programs to be uploaded regularly in non-volatile memory and eliminated the need for large amounts of memory to be required onboard.
>
> The original software for the Voyager probes was written using Fortran 5 then ported to Fortran 77, and today there is some porting in C. Low-level, lightweight software is increasingly important as the probes move farther and farther away from Earth and communication becomes slower.

I hope the reader will indulge me in proposing a tangential perspective on "Cosmos". Because this book is based on an uncanny premise: a perspective on our life as a species seen from space, as if an alien life form had authored it after studying us closely for a while.

The sixth chapter of Cosmos, called "Travelers' Tales", tells precisely the story of the Voyager probes. It starts by telling the story of Dutch explorers in the seventeenth century, then the story of renowned scientists and explorers like Galileo[7], Huygens[8], Leeuwenhoek[9], and Zheng He[10]. In this sixth chapter, Sagan describes a logbook, where an imaginary pilot inside the Voyager probes would write down their experience day after day, as they move farther and farther away from planet Earth. Just like seventeenth century travelers using stars to find their way across the oceans, the Voyagers quite literally still do the same.

According to Sagan, we are just "One Voice in the Cosmic Fugue" (chapter 2) living at "The Shores of the Cosmic Ocean" (chapter 1). And when he asked, "Who Speaks for Earth?" (chapter 13) we stayed shamefully silent.

The idea of an "Encyclopædia Galactica" (chapter 12) sent my mind on overdrive. Is there one? Is there a file about us somewhere in it? Jodie Foster's character, in this clumsy adaptation[11] of yet another book by Carl Sagan, also briefly mentioned it when discussing the incoming signal discovered by SETI[12]. I guess Wikipedia is the closest thing to an Encyclopædia Galactica that I will get to see during my lifetime on this planet.

Sprinkled inside the pages of "Cosmos" there are dozens of gorgeous photos, taken by the onboard cameras of various spacecraft, often processed in "false color" and igniting my mind with the most incredible views of worlds far away. Another famous photograph, understandably not featured in the pages of "Cosmos", was the one known as "Pale Blue Dot"[13], taken

[6] https://www.allaboutcircuits.com/news/voyager-mission-anniversary-computers-command-data-attitude-control/

[7] https://en.wikipedia.org/wiki/Galileo_Galilei

[8] https://en.wikipedia.org/wiki/Christiaan_Huygens

[9] https://en.wikipedia.org/wiki/Antonie_van_Leeuwenhoek

[10] https://en.wikipedia.org/wiki/Zheng_He

[11] https://en.wikipedia.org/wiki/Contact_(1997_American_film)

[12] https://en.wikipedia.org/wiki/Search_for_extraterrestrial_intelligence

[13] https://en.wikipedia.org/wiki/Pale_Blue_Dot

by Voyager 1 in 1990. This photograph inspired Carl Sagan yet another book[14].

Cosmos is the first science book I read as a kid. I had watched (needless to say, in absolute awe) the TV series[15] around 1980, but one day, walking around Buenos Aires, I discovered the companion book on display in one of the city's largest bookstores. I begged my mother to buy me a copy. This book is still with me, almost 40 years later and after three migrations across the Atlantic, and I must have read it a dozen times. It is challenging to put in words the impact that this book has had on me.

As Bruce Lewenstein said in 2002[16],

> Ultimately, books create the culture that we live in. They are elements both of the scientific culture and of our more general culture. By looking at them we can actually see the ways in which science and modern culture are not separate but are — to use a jargon word from the sociology of science — co-produced. Neither science nor society exists without the other one.

As a side effect of my fanaticism for the "Cosmos" TV series, I became obsessed with Vangelis[17]' music, and of course, I bought the cassette tape[18] with the official soundtrack, but that is another story. Today, in our age of streaming services, you can listen to it on Spotify[19].

The Voyager probes have seemingly been there my whole life. Voyager 1 was launched the day after I celebrated my 4th birthday. I watched the TV series on Canal 13[20] when I was in first grade, more or less at the same time when U2[21] and Depeche Mode[22] were recording their first albums. I saw Voyager 2's pictures of Uranus in the pages of the Argentine edition of the "Muy Interesante"[23] magazine as I was heading towards high school. Years later, I read Sagan's "Dragons of Eden" and "Pale Blue Dot" books while studying physics in college. Carl Sagan passed away shortly before I began a career in software engineering.

Decades later, I got news of the Voyagers crossing the heliopause while I was busy making mobile apps for a living. In 2016, I bought a copy of the re-edition of the Voyager Golden Record[24] on Kickstarter, used by movies such as Star Trek: The Motion Picture[25] (1979) and John Carpenter's Starman[26] (1984) as a basis for their plots. And now, as I enter my sixth decade on Earth, writing a monthly magazine about computer programming, boom; they show up in the news once again, receiving a remarkable software update, still phoning home every so often.

We are living in a truly remarkable age. Godspeed, Voyagers.

Cover photo by the author.

---

[14] https://en.wikipedia.org/wiki/Pale_Blue_Dot_(book)
[15] https://en.wikipedia.org/wiki/Cosmos%3A_A_Personal_Voyage
[16] https://web.archive.org/web/20070921181105/https://www.nist.gov/public_affairs/bestpractices/Lewenstein2.htm
[17] https://en.wikipedia.org/wiki/Vangelis
[18] https://en.wikipedia.org/wiki/Cassette_tape
[19] https://open.spotify.com/playlist/5DiaWEtyusrATyvF7uzJDJ?si=c5ed6fa140844f91
[20] https://en.wikipedia.org/wiki/El_Trece
[21] https://en.wikipedia.org/wiki/U2
[22] https://en.wikipedia.org/wiki/Depeche_Mode
[23] https://en.wikipedia.org/wiki/Muy_Interesante
[24] https://www.kickstarter.com/projects/ozmarecords/voyager-golden-record-40th-anniversary-edition
[25] https://en.wikipedia.org/wiki/Star_Trek:_The_Motion_Picture
[26] https://en.wikipedia.org/wiki/Starman_(film)