Falsehoods Programmers Believe About Time

A Preprint

Noah Sussman

September 5, 2025
DOI:10.5281/zenodo.17070518

Abstract

This paper compiles and presents two previously published essays, Falsehoods Programmers Believe About Time *and* More Falsehoods Programmers Believe About Time: "Wisdom of the Crowd" Edition*. Originally circulated online in 2012, they are reproduced here in full and without modification. By consolidating these essays in a single paper, the goal is to provide a stable, citable resource for research and practice.*

The work systematizes common misconceptions about temporal modeling in computational systems. It characterizes difficult cases such as temporal boundaries at day-and-year-end transitions, irregularities in calendar cycles, distributed clock drift, and jurisdiction-dependent variability in time zone definitions. The result is a structured taxonomy of temporal edge cases designed to support more reliable and reproducible time-aware applications.

Keywords AI Safety · Corner Case Testing · Data Quality · Distributed Computing · Failure Modes · Gray Literature · Practitioner Knowledge · Software Engineering · System Reliability · Temporal Data · Temporal Representation · Temporal Semantics · Time Zones

Introduction

Software engineers are often surprised by the complexity of temporal representation in computational systems. In one observed incident, a defect in the KVM virtual machine platform caused suspended instances to stop updating their clocks, producing anomalous behavior that was not immediately attributable. The unusual severity of this defect motivated the author to maintain a journal of time-related errors, which subsequently formed the basis for the original essay.

The second entry in the *Falsehoods about X* genre, the essay was featured in *WIRED* and *BoingBoing*, and Jeff Atwood referred to it as "classic." [1] More than 500 combined comments on Hacker News, Reddit, and Slashdot amplified its visibility, and it subsequently appeared in Stack Overflow and Unix & Linux Stack Exchange discussions as a canonical reference for programmers grappling with time and date arithmetic (for example, "get previous month regardless of days").

The work has circulated internationally, with unofficial translations available in both Russian [4] and Mandarin [5].

Over time, it has become a canonical reference for both practitioners and researchers. Excerpts have been incorporated into introductory computer science curricula [2], [3], and the work has been cited in publications across Media Studies, Digital Theatre, Design Ethics, Anthropology, Software Architecture, and AI Code Generation. Across these disciplines, it provides foundational evidence that temporal representation is sociotechnical in nature and inherently complex.

By consolidating observed misconceptions into a taxonomy, the original work bridged practitioner experience with scholarly analysis. It continues to serve as a stable reference point for pedagogy,

empirical research, system design, and the emerging discourse on AI safety, where temporal misalignment is treated as a critical risk factor.

Falsehoods Programmers Believe About Time

Over the past couple of years I have spent a lot of time debugging other engineers' test code. This was interesting work, occasionally frustrating but always informative. One might not immediately think that test code would have bugs, but of course all code has bugs and tests are no exception.

I have repeatedly been confounded to discover just how many mistakes in both test and application code stem from misunderstandings or misconceptions about time. By this I mean both the interesting way in which computers handle time, and the fundamental gotchas inherent in how we humans have constructed our calendar — daylight savings being just the tip of the iceberg.

In fact I have seen so many of these misconceptions crop up in other people's (and my own) programs that I thought it would be worthwhile to collect a list of the more common problems here.

All of these assumptions are wrong

1. There are always 24 hours in a day.
2. Months have either 30 or 31 days.
3. Years have 365 days.
4. February is always 28 days long.
5. Any 24-hour period will always begin and end in the same day (or week, or month).
6. A week always begins and ends in the same month.
7. A week (or a month) always begins and ends in the same year[1] [6].
8. The machine that a program runs on will always be in the GMT time zone.
9. Ok, that's not true. But at least the time zone in which a program has to run will never change.
10. Well, surely there will never be a change to the time zone in which a program hast to run in production.
11. The system clock will always be set to the correct local time.
12. The system clock will always be set to a time that is not wildly different from the correct local time.
13. If the system clock is incorrect, it will at least always be off by a consistent number of seconds.
14. The server clock and the client clock will always be set to the same time.
15. The server clock and the client clock will always be set to around the same time.
16. Ok, but the time on the server clock and time on the client clock would never be different by a matter of decades.
17. If the server clock and the client clock are not in synch, they will at least always be out of synch by a consistent number of seconds.
18. The server clock and the client clock will use the same time zone.
19. The system clock will never be set to a time that is in the distant past or the far future.
20. Time has no beginning and no end [7].
21. One minute on the system clock has exactly the same duration as one minute on any other clock[2] [8].

[1] This statement confused a lot of people. What I originally meant was that a month-long period may not end in the same year that it started. However, see the Reddit comment thread for discussion of several special cases where New Years Day falls in the middle of a month

[2] Yes, there's a very rigorous standard for the duration of units of time such as seconds and minutes. But no, your system clock probably doesn't have any *direct* knowledge of that standard.

22. Ok, but the duration of one minute on the system clock will be pretty close to the duration of one minute on most other clocks.
23. Fine, but the duration of one minute on the system clock would never be more than an hour.
24. You can't be serious.
25. The smallest unit of time is one second.
26. Ok, one millisecond.
27. It will never be necessary to set the system time to any value other than the correct local time.
28. Ok, testing might require setting the system time to a value other than the correct local time but it will never be necessary to do so in production.
29. Time stamps will always be specified in a commonly-understood format like `1339972628` or `133997262837`.
30. Time stamps will always be specified in the same format.
31. Time stamps will always have the same level of precision.
32. A time stamp of sufficient precision can safely be considered unique.
33. A timestamp represents the time that an event actually occurred.
34. Human-readable dates can be specified in universally understood formats such as `05/07/11`.

That thing about a minute being longer than an hour was a joke, right?

No.

There was a fascinating bug in older versions of KVM [9] on CentOS. Specifically, a KVM virtual machine had no awareness that it was not running on physical hardware. This meant that if the host OS put the VM into a suspended state, the virtualized system clock would retain the time that it had had when it was suspended. E.g. if the VM was suspended at 13:00 and then brought back to an active state two hours later (at 15:00), the system clock on the VM would still reflect a local time of 13:00. The result was that every time a KVM VM went idle, the host OS would put it into a suspended state and the VM's system clock would start to drift away from reality, sometimes by a large margin depending on how long the VM had remained idle.

There was a cron job that could be installed to keep the virtualized system clock in line with the host OS's hardware clock. But it was easy to forget to do this on new VMs and failure to do so led to much hilarity. The bug has been fixed in more recent versions.

An acknowledgment

This post owes a great debt to Patrick McKenzie's canonical blog post about user names [10], which I have read over and over throughout the years and from which I have shamelessly cribbed both concept and style. If you haven't yet read this gem, go and do so right now. I promise you'll enjoy it.

Updates and Community Contributions

I'd like to say thanks to everyone who contributed to the comment threads about this post on BoingBoing [11], Hacker News [13], Reddit [14], MetaFilter [15] and to everyone on Twitter who shared their strange experiences with time.

You have provided so many interesting edge cases I had forgotten about as well as many oddities of which I wasn't aware. For instance: in the Jewish calendar, days start at sunset not midnight [16]. And as Bruce Sterling [19] pointed out, I didn't even think about what happens when the computer is on a spaceship orbiting a black hole.

There's more than enough material for another (longer!) post about this topic. But first I'll have to finish reading all >500 of your comments as well as the wealth of awesome research material[3] [12] that has been linked.

[3]Thanks to HN user snprbob86 for pointing me to 'The Long, Painful History of Time' by Erik Naggum. Looks great, can't wait to read it

I've written another post collecting the many other falsehoods that were suggested by your comments at various platforms. Thanks again for your enthusiasm and for the mind-boggling level of detail. I learned a lot about time in the last 24 hours. Fellow nerds, I salute you.

Canonical URL

Falsehoods Programmers Believe About Time now has a canonical permalink you may use when referring to this post: `FalsehoodsAboutTime.com`

More Falsehoods Programmers Believe About Time: "Wisdom of the Crowd" Edition

A couple of days ago I decided to write down some of the things I've learned about testing over the course of the last several years. In the course of enumerating the areas that benefit most from testing, I realized that I had accumulated a lot of specific thoughts about how we as programmers tend to abuse the concept of *time*.

So I wrote another post called **"falsehoods programmers believe about time,"** where I included 34 misconceptions and mistakes having to do with both calendar and system time. Most of these were drawn from my immediate experience with code that needed to be debugged (both in production and in test).

A great many of the false assumptions listed were my own. Especially "time stamps are always in seconds since epoch" and "the duration of a system clock minute is always pretty close to the duration of a wall clock minute." Whoa did I ever live to regret my ignorance in those two cases! But hey, apparently I'm not the only one who has run into (or inadvertently caused) such issues. A lot of people responded and shared similar experiences.

I'd like to say an enormous thanks to everyone who contributed to the comment threads on BoingBoing [11], Hacker News [13], Reddit [14], MetaFilter [15] and to everyone on Twitter who shared their strange experiences with time. In those thousand or so comments and tweets, there were a lot of suggestions as to "falsehoods 35 to $35 + n$."

First and foremost was the omission of the false assumption that "time always moves forward," as pointed out by Technomancy and many others.

I enjoyed reading all the suggested falsehoods. When I was done reading, I realized that taken as a whole, these constitute a whole other blog post. So I collected some of your suggested falsehoods into a post and here it is.

All of these assumptions are wrong

All of these falsehoods were suggested by people who commented on the original post. Each contributor is credited below.

1. The offsets between two time zones will remain constant.
2. OK, historical oddities aside, the offsets between two time zones won't change in the future.
3. Changes in the offsets between time zones will occur with plenty of advance notice.
4. Daylight saving time happens at the same time every year.
5. Daylight saving time happens at the same time in every time zone.
6. Daylight saving time always adjusts by an hour.
7. Months have either 28, 29, 30, or 31 days.
8. The day of the month always advances contiguously from n to either $n + 1$ or 1, with no discontinuities.
9. There is only one calendar system in use at one time.
10. There is a leap year every year divisible by 4 [17].
11. Non leap years will never contain a leap day.

12. It will be easy to calculate the duration of x number of hours and minutes from a particular point in time.
13. The same month has the same number of days in it everywhere!
14. Unix time is completely ignorant about anything except seconds.
15. Unix time is the number of seconds since Jan 1st 1970.
16. The day before Saturday is always Friday.
17. Contiguous timezones are no more than an hour apart.
18. Two timezones that differ will differ by an integer number of half hours.
19. Okay, quarter hours.
20. Okay, seconds, but it will be a consistent difference if we ignore DST.
21. If you create two date objects right beside each other, they'll represent the same time.
22. You can wait for the clock to reach exactly `HH:MM:SS` by sampling once a second.
23. If a process runs for n seconds and then terminates, approximately n seconds will have elapsed on the system clock at the time of termination.
24. Weeks start on Monday.
25. Days begin in the morning.
26. Holidays span an integer number of whole days.
27. The weekend consists of Saturday and Sunday.
28. It's possible to establish a total ordering on timestamps that is useful outside your system.
29. The local time offset (from UTC) will not change during office hours.
30. `Thread.sleep(1000)` sleeps for 1000 milliseconds.
31. `Thread.sleep(1000)` sleeps for $>= 1000$ milliseconds.
32. There are 60 seconds in every minute.
33. Timestamps always advance monotonically [18].
34. GMT and UTC are the same timezone.
35. Britain uses GMT.
36. Time always goes forwards.
37. The difference between the current time and one week from the current time is always `7 * 86400` seconds.
38. The difference between two timestamps is an accurate measure of the time that elapsed between them.
39. `24:12:34` is an invalid time.
40. Every integer is a theoretical possible year.
41. If you display a datetime, the displayed time has the same second part as the stored time.
42. Or the same year.
43. But at least the numerical difference between the displayed and stored year will be less than 2.
44. If you have a date in a correct YYYY-MM-DD format, the year consists of four characters.
45. If you merge two dates, by taking the month from the first and the day/year from the second, you get a valid date.
46. But it will work, if both years are leap years.
47. If you take a w3c published algorithm for adding durations to dates, it will work in all cases.
48. The standard library supports negative years and years above 10000.
49. Time zones always differ by a whole hour.
50. If you convert a timestamp with millisecond precision to a date time with second precision, you can safely ignore the millisecond fractions.

51. But you can ignore the millisecond fraction, if it is less than 0.5.
52. Two-digit years should be somewhere in the range 1900–2099.
53. If you parse a date time, you can read the numbers character for character, without needing to backtrack.
54. But if you print a date time, you can write the numbers character for character, without needing to backtrack.
55. You will never have to parse a format like `---12Z` or `P12Y34M56DT78H90M12.345S`.
56. There are only 24 time zones.
57. Time zones are always whole hours away from UTC.
58. Daylight Saving Time (DST) starts/ends on the same date everywhere.
59. DST is always an advancement by 1 hour.
60. Reading the client's clock and comparing to UTC is a good way to determine their timezone.
61. The software stack will/won't try to automatically adjust for timezone/DST.
62. My software is only used internally/locally, so I don't have to worry about timezones.
63. My software stack will handle it without me needing to do anything special.
64. I can easily maintain a timezone list myself.
65. All measurements of time on a given clock will occur within the same frame of reference [6].
66. The fact that a date-based function works now means it will work on any date.
67. Years have 365 or 366 days.
68. Each calendar date is followed by the next in sequence, without skipping.
69. A given date and/or time unambiguously identifies a unique moment.
70. Leap years occur every 4 years.
71. You can determine the time zone from the state/province.
72. You can determine the time zone from the city/town.
73. Time passes at the same speed on top of a mountain and at the bottom of a valley.
74. One hour is as long as the next in all time systems.
75. You can calculate when leap seconds will be added.
76. The precision of the data type returned by a `getCurrentTime()` function is the same as the precision of that function.
77. Two subsequent calls to a `getCurrentTime()` function will return distinct results.
78. The second of two subsequent calls to a `getCurrentTime()` function will return a larger result.
79. The software will never run on a space ship that is orbiting a black hole.

Seriously? Black holes?

Hey, if Bruce Sterling [19] says that my software needs to be resilient against time distortions caused by black holes, I'm going to take him at his word.

Corrections

Daniel Morrison pointed out that it's daylight saving time and not daylight savings time. Thanks!

Credits

Thanks again to everyone who commented. I read everything that you wrote, even if I didn't wind up including it above.

I made the list above by going through each of the comment threads on Hacker News [13], Reddit [14], MetaFilter [15] and BoingBoing [11] (in that order) and finding all(?) of the places where folks had broken out "falsehood 35 to $35 + n$" as a bulleted list. I then selectively copied those lists — in the order that I found them. I made small edits for readability and occasionally I paraphrased (this is noted below).

From Hacker News

1–8: JoshTriplett, 9–10: lambda, 11: hc5, 12: chris_wot, 13: einhverfr, 14: masklinn, 15: rmc, 16: jimfl, 17: einhverfr, 18–20: aardvark179, 21–22: bazzargh, 23: my paraphrase of mikeash's comment, 24–26: edanm, 27: my paraphrase of Mvandenbergh's comment, 28: derleth, 29: finnw, 30: michaelochurch, 31: cpeterso, 32–33: dfranke, 34: arohner, 35: TazeTSchnitzel, 36: technomancy, 37: sses, 38: DanWaterworth

From Reddit

39–55: benibela2, 56–64: Darkhack, 65: ericanderton, 66: Taladar

From MetaFilter

67–69: Joe in Australia

From BoingBoing

70–75: Paul

From Twitter

76–79: cmchen

An acknowledgment

This post — like the one before it — owes a great debt to Patrick McKenzie's canonical blog post about user names [10], which I have read over and over throughout the years and from which I have shamelessly cribbed both concept and style. If you haven't yet read this gem, go and do so right now. I promise you'll enjoy it.

References

[1] Atwood, Jeff. Doing Terrible Things to Your Code. *Coding Horror*, 2012. https://blog.codinghorror.com/doing-terrible-things-to-your-code/.

[2] Jones, Matthew J. Lecture 11: Datetime myths. Course lecture, Graduate Center, City University of New York, 2023. https://www.math.csi.cuny.edu/~mvj/GC-DataViz-S23/lectures/L11.html.

[3] University of Washington Department of Computer Science and Engineering. CSE143 style guide: Avoid anglocentrism. Course style guide, University of Washington, 2016. https://courses.cs.washington.edu/courses/cse143/16au/style/avoid-anglocentrism.html.

[4] Sussman, Noah. Заблуждения программистов относительно времени. *Habr*, 2012. https://habr.com/ru/articles/146109/.

[5] Huli. 淺談 JavaScript 中的時間與時區處理. *Blog Huli*, 2020. https://blog.huli.tw/2020/12/26/javascript-date-time-and-timezone/.

[6] Anonymous. All measurements of time on a given clock. Reddit comment, 2012. http://www.reddit.com/r/programming/comments/v8s0y/falsehoods_programmers_believe_about_time/c52kqpa.

[7] Wikipedia contributors. Year 2038 problem. *Wikipedia, the free encyclopedia*, 2012. https://en.wikipedia.org/wiki/Year_2038_problem.

[8] Wikipedia contributors. Atomic clock. *Wikipedia, the free encyclopedia*, 2012. https://en.wikipedia.org/wiki/Atomic_clock.

[9] Wikipedia contributors. Kernel-based virtual machine. *Wikipedia, the free encyclopedia*, 2012. https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine.

[10] McKenzie, Patrick. Falsehoods programmers believe about names. *Kalzumeus Software*, 2010. https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/.

[11] Doctorow, Corey. Falsehoods programmers believe about time. *BoingBoing*, 2012. https://boingboing.net/2012/06/19/falsehoods-programmers-believe.html.

[12] Naggum, Erik. The long, painful history of time. *Naggum.no*, 1999. https://naggum.no/lugm-time.html.

[13] Hacker News contributors. Falsehoods programmers believe about time. Discussion thread, 2012. https://news.ycombinator.com/item?id=4128208.

[14] Reddit contributors. Falsehoods programmers believe about time. Discussion thread, 2012. http://www.reddit.com/r/programming/comments/v8s0y/falsehoods_programmers_believe_about_time/.

[15] MetaFilter contributors. Falsehoods programmers believe about time. Discussion thread, 2012. http://www.metafilter.com/117073/Falsehoods-Programmers-Believe.

[16] Anonymous. Falsehoods programmers believe about time. Hacker News comment, 2012. https://news.ycombinator.com/item?id=4130904.

[17] Microsoft Corporation. Knowledge base article 214326: Leap year rule. Technical documentation. https://support.microsoft.com/kb/214326.

[18] Anonymous. Timestamps always advance monotonically. MetaFilter comment, 2012. http://www.metafilter.com/117073/Falsehoods-Programmers-Believe#4405410.

[19] Sterling, Bruce. Falsehoods programmers believe about time. *WIRED*, 2012. https://web.archive.org/web/20120622003551/https://www.wired.com/beyond_the_beyond/2012/06/falsehoods-programmers-believe-about-time/.