AT&T Long-Distance Network Crash: Israel Okuneye

A crash unprecedented in the history of telecommunications took place on Monday, January 15, 1990. Network managers at AT&T started noticing red alert signals from various parts of their world-wide network. A huge portion of the network reported the same warning in a couple of seconds, which resulted in a domino effect from one computer site to another. This paper explains the background and causes, as well as aftermath and current status of the crash. The ethical considerations of AT&T – the way it responded to its subscribers, and the significance of the crash are explained in detail.

Background and Causes

The problem was caused by a computer bug in the latest update by AT&T that impacted 4ESS long distance switches – equipment that processed signals. Every computer program is tested over and again before being pushed into production. However, when new software is developed, errors are likely to occur between the old and new software. According to Neumann, "The fault was in the code of the new software that AT&T loaded into front-end processors of all 114 of its 4ESS switching systems in mid-December, said Larry Seese, AT&T's director of technology development" (1990, para. 1). Figure 1 shows the pseudocode of the defective code.

Figure 1

In Pseudocode, the Program Read as Follows (Burke, 1995)

```
1  while (ring receive buffer not empty
          and side buffer not empty) DO

2     Initialize pointer to first message in side buffer
      or ring receive buffer

3     get copy of buffer

4     switch (message)

5        case (incoming_message):

6              if (sending switch is out of service) DO

7                 if (ring write buffer is empty) DO

8                    send "in service" to status map

9                 else

10                   break

                  END IF

11             process incoming message, set up pointers to
               optional parameters

12             break
       END SWITCH


13    do optional parameter work
```

The failure stemmed from a signaling system in New York City that informed the network of which of several alternative routes a telephone call will take. The system appeared to have sent a message to the national network claiming that all circuits were busy when they were not. In a circumstance like this one, the system was meant to conduct corrective initialization that only lasts between 4 and 6 seconds, which it did, then went back into service and began processing calls again. Neumann stated that,

> Under the previous system, switch A would send out a message that it was working again, and switch B would double-check that switch A was back in service. With the new software, switch A begins processing calls and sends out call routing signals. The reappearance of traffic from switch A is supposed to tell switch B that A is working again. (1990, para. 3)

Put simply, switch A was supposed to inform switch B that it can now route calls again.

It resulted in a crash-reboot cycle of all 114 4ESS switching systems every 6 seconds, blocking over 50 million calls in 9 hours. Approximately 100 AT&T engineers were eventually able to detect and correct the bug which stopped the crash-reboot cycle and restored service (Burke, 1995).

Aftermath and Current Status

The problem was rectified by AT&T decreasing the messaging load of the CCS7 network. The switches were able to relax and the network stabilized (Neumann, 1990). Decreasing the load on the switches was an effective way of making sure that they performed efficiently and also minimized the chances of crashing again.

AT&T announced around midnight that the problem appeared to have been resolved and that the switching system had been operational since 11:30 p.m. that day (Sims, 1990). Sims (1990) reported that Walter Murphy, AT&T spokesman said "'We have put in place some software fixes that stabilized the network and contained the problem. The root cause of the problem is still to be investigated and determined, but the network should process calls normally today'" (para. 4).

Nearly half of AT&T's calls were unsuccessful. The unconnected calls cost AT&T more than $60 million. Many AT&T-dependent businesses, such as airlines, hotels, and car rental services, were unable to operate normally (Burke, 1995).

Ethical Considerations

AT&T operators did not act ethically when they refused to tell subscribers they could dial access codes to place their calls using other long-distance carriers (Sims, 1990). AT&T acted in its own best interest, which is psychological egoism whereas, it could have acted honestly and in the best interest of its subscribers by giving them these access codes. It would have earned the respect and admiration of its subscribers if it had acted differently. According to Sims, “The operators said they did not know the access numbers, and that it was against company policy to provide information on a competitor” (1990, para. 9).

AT&T eventually reached out to customers and apologized for the breakdown. Robert E. Allen, AT&T Chairman, admitted that operators exacerbated the problem by refusing to provide subscribers the access codes required to access competitors' long-distance systems (Tumulty, 1990). Providing these access codes would have minimized the effect of the crash on its affected subscribers, especially those who are AT&T-dependent in their day-to-day transactions.

Significance and Impact

Some of the key significances of this network crash are that software testing and the use of better structured programs cannot be overemphasized. Burke stated that,

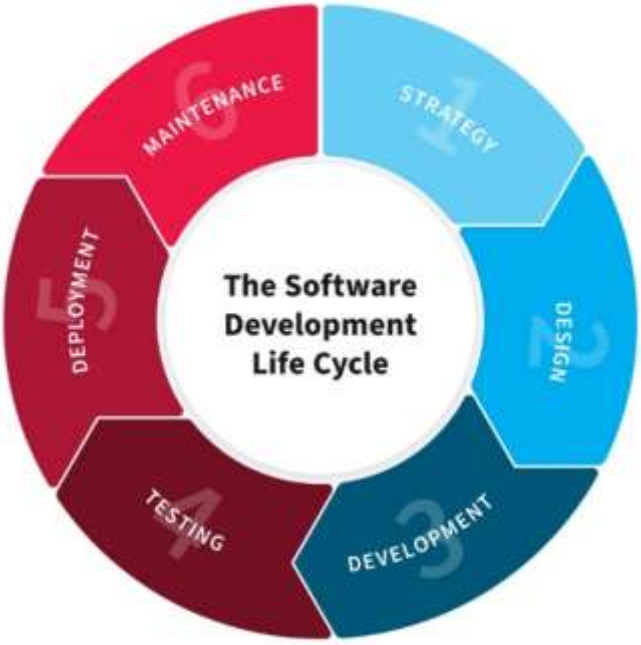
> There is still much to be learned from this incident, however. Clearly, the use of C programs and compilers contributed to the breakdown. A more structured programming language with stricter compilers would have made this particular defect much more obvious. The routine practice of allowing the long-distance switches to shut down and reset themselves also contributed. A more fault-tolerant hardware and software system that could handle minor problems without

shutting down could have greatly reduced the effects of the defect. (1995, para. 7)

According to Burke, “The software update loaded in the 4ESSs had already passed through layers of testing and had remained unnoticed through the busy Christmas season. AT&T was fanatical about its reliability” (1995, para. 6). Figure 2 shows the phases of software development life cycle.

Figure 2

The Software Development Life Cycle (Vorobiova, 2021)



The software development life cycle requires that a system goes through multiple test phases to demonstrate a working product. Modules are first tested independently, and once confirmed to work seamlessly, integration testing then focuses on dependencies between modules. It is worth noting that, despite AT&T's careful attention to hardware survivability and extensive testing, this is one of the few problems that has ever had such a significant impact on its long-distance network (Burke, 1995). Companies can use this case to ensure that their software engineers devote more time and effort to software testing. This will keep them out of a similar situation in the future.

References

Burke, D. (1995). *All Circuits are Busy Now: The 1990 AT&T Network Collapse.* California Polytechnic State University. https://users.csc.calpoly.edu/~jdalbey/SWE/Papers/att_collapse

Neumann, G. P. (1990). *The Crash of the AT&T Network in 1990.* Telephone World. https://telephoneworld.org/landline-telephone-history/the-crash-of-the-att-network-in-1990/

Sims, C. (1990). *Computer Failure Disrupts AT&T Long Distance.* The New York Times. https://www.nytimes.com/1990/01/16/us/computer-failure-disrupts-at-t-long-distance.html

Tumulty, K. (1990). AT&T; Reaches Out to Public and Apologizes for Breakdown: Telecommunications: The firm suspects that a 'bug' in computer software caused its system's collapse. It may offer restitution to some and a day of discounted rates to all. *Los Angeles Times.* https://www.latimes.com/archives/la-xpm-1990-01-17-fi-215-story.html

Vorobiova, A. (2021). *Software Development Life Cycle: NIX Approach to SDLC.* NIX. https://nix-united.com/blog/software-development-life-cycle-nix-approach-to-sdlc/