NATIONAL
PREPAREDNESS
COMMISSION

Lessons from the Millennium Bug

Martyn Thomas CBE FREng

February 2021

This paper has been prepared for the National Preparedness Commission. It is based on a paper that accompanied a Gresham College lecture[i]. Please refer to the transcript[ii] of that lecture for considerably more detail and supporting references.

CONTENTS

EXECUTIVE SUMMARY

The Millennium Bug or Y2K issue arose when it was feared that computers would fail at midnight on 1 January 2000. The concerns related to the formatting and storage of data for dates beginning in 2000 because many computer programs represented four-digit years with only the final two digits, making the year 2000 indistinguishable from 1900.

This paper explains why Y2K was potentially so serious and how the risks were addressed by co-ordinated and timely action to mitigate them. It also points to important lessons from that experience to ensure future resilience.

The principal lessons of the Y2K experience that are relevant for the future are:

- **Go beyond testing.** Even though testing is still the primary way in which programmers assure that software is fit for purpose, testing can never find all the errors in IT systems. It is necessary to move away from the idea that cost and time-to-market take priority over modularity, robustness, security, ease of modification and other software engineering principles.
- **Reduce cascading effects.** Shared points of failure are still introduced without considering the possible consequences of cascading effects. Elevating potential risks to Board level, as auditors did for Y2K, can help with prioritisation and resourcing.
- **Supply-chains redundancy.** In the interests of efficiency, supply chains have become far more tightly coupled and redundancy has been removed with little thought about the impact on resilience, again making cascade failures more likely. This needs to be addressed when introducing major IT systems.
- **Better regulation.** There is no government appetite for using regulation to encourage private companies to write better software. Since the 1980s, industry has successfully argued that software is not a product but a service and that it should therefore be exempt from product safety and consumer protection laws. Such exemption should be reviewed.

Background

The core concern was that IT programs would not recognise 21st Century dates properly and that this would cause various errors such as the incorrect display of dates and the inaccurate ordering of automated, dated records or real-time events. There were general fears that this could result in power blackouts, planes failing in flight, and computer records of bank accounts disappearing, simultaneously wiping out both savings and debts. Y2K was considered a particularly serious threat because it could cause many unrelated systems to fail simultaneously.

Much time, effort and resources were expended by governments and businesses alike to ensure – by and large successfully – that the worst did not happen. The total cost of remediation was estimated at between US$300bn and US$500bn. This would, however, have been dwarfed by the consequences of power blackouts, transport failing, bank records wiped, etc. As it was, many potential points of failure were identified as part of the mitigation measures and corrected. Many of these were caused by poor engineering software that could cause unrelated systems to fail simultaneously.

Conclusions

Contrary to popular belief that experts misjudged, and exaggerated, the scale of the problem, remediation was largely successful precisely because those expert warnings were heard and acted upon. Y2K should be regarded as a signal event – a near miss that should serve as one more major stimulus to change the way that IT systems are designed and built, so that no similar problem can happen again. But there are also wider lessons for national preparedness.

Y2K remediation showed that it is possible to deliver major IT system resilience on time given clear objectives, clear timescales, senior leadership attention, a commitment to provide the necessary resources, clear communications, and acceptance throughout the potentially affected organisations that the challenge is real and deserves the highest priority.

One saving grace of the Y2K effort was the fact that IT systems did not prove to be as tightly coupled as had been feared. Yet, systems have become still more complex and inter-dependent since 2000 making decoupling and redundancy important in removing the risk of cascading failures. Can we be sure that there is no potentially catastrophic 'Bug' lurking somewhere in that system, and are we sufficiently prepared to manage to manage the consequences if there is?

Aim: This paper explains why Y2K was potentially so serious and how the risks were addressed by co-ordinated and timely action to mitigate them. It also points to important lessons from that experience to ensure future resilience.

INTRODUCTION

During the 1990s, newspapers contained a growing number of stories and warnings about the Century Date Problem or 'Millennium Bug' (Y2K) that could cause computers to fail at midnight on 1 January 2000. There were fears that there could be a power blackout and failure of the water supply, and that computer records of bank accounts could disappear, simultaneously wiping out both savings and debts. Governments worried about safety and security, auditors questioned whether they could approve company accounts if companies collapsed, airlines and passengers did not know whether planes would fail in flight or whether airports would be able to remain open and radars and air traffic control would continue to function.

Committees were formed and issued reports and advice. Auditors insisted that companies obtained assurances about their essential systems or replaced them. New consultancy companies were created to investigate systems and equipment, to advise on Y2K risks, and to manage Y2K projects. Old programmers were brought out of retirement to look again at the systems they and their colleagues had written years or decades before. For a few years it seemed that the launch of the European common currency, the Euro, may have to be delayed from the planned date of 1 January 1999.

As the new millennium approached, there was growing anxiety in boardrooms, among technical staff and across many parts of society in the UK and other advanced economies. Despite the huge expenditure and efforts, it was difficult to be sure that nothing vital had been overlooked. Software errors were a familiar experience, of course. What made this different is that it could potentially cause very many systems to fail at the same time.

Then nothing happened or so it seemed and the feeling grew that the whole thing had been a myth or invented by rapacious consultants, supported by manufacturers who wanted to compel their customers to throw away perfectly good equipment and buy the latest versions.

There is a prevention paradox that means that important lessons are rarely learned from near misses from calamities that were avoided by chance or by sustained hard work. Y2K has entered popular culture as the main example of a problem that was grossly exaggerated and that was never serious, although in reality it was a major crisis where catastrophes were only prevented by thousands-of-person years of effort backed by international co-operation.

THE PROBLEM

In 1990, most computing systems contained serious errors that would cause failure when they encountered dates in 2000 or later. Most business data-processing systems had multiple Y2K errors. So did many microprocessor-based 'embedded systems' that displayed the current date, calculated someone's age from their date of birth, checked expiry dates for security passes, credit cards, food or medicines, generated dated printouts, carried out specific functions on specific days, or stored information for averaging or trending purposes. This could affect a very wide range of systems, such as personal computers, surveillance equipment, lighting systems, entry systems, barcode systems, vending machines, switchboards, safes and time locks, lifts (they maintain dates for maintenance and alarms), faxes, vehicles, process monitoring systems, and production line equipment.

The risks were slowly recognised to be serious and widespread. A UK Government report in 1996 said that the number of embedded systems distributed worldwide was around 7bn and that around 5% of embedded systems were found to fail millennium compliance tests. More sophisticated embedded systems had failure rates of between 50% and 80%. In manufacturing environments, an overall failure rate of around 15% was typical[iii].

Companies had to start by preparing an inventory of all their important systems. This was when the scale of their risks and of the work they faced became apparent. For most of them, this was the largest IT project they had ever had to undertake and they lacked the technical and management skills. Even FT100 companies had poor software management systems, few could find all the source code for their business-critical systems because changes had been made in private libraries and without updating documentation. They also faced an urgent and immovable deadline as systems would start to fail as soon as they encountered dates in 2000 or later. An early failure occurred in 1987 when Marks and Spencer received a batch of tinned meat that had a use-by date of January 2000 (01/00) which the goods-inbound system interpreted as January 1900 – making the meat 87 years too old.

There were several related problems to find and solve:

- The processing of two-digit years.
- Real-time clocks in PCs, Programmable Logic Controllers and other microprocessor-based systems in equipment, buildings, factories, vehicles, nationally and internationally.
- The first leap-year since 1600 that was a century year because century years are leap years only if they divide by 400.
- Special uses of dates (99 and 00 were widely used for special purposes, such as marking the end of a set of records or to mark data that had already been processed).
- Quite a lot of systems had the century built into data or in a programme as a constant '19'. Many organisations had '19' pre-printed on forms, contracts, purchase orders, invoices and cheques. All these forms and pro-forma data fields had to be changed for the new century – often just for reasons of clarity and professional appearance but sometimes because giving the wrong date could have legal consequences such as making a cheque or other legal document invalid.

Many companies only recognised the risks they faced when, in the mid-1990s, the major audit firms started to tell all their clients that they would be unable to audit accounts on a going concern basis in 1998 unless there was evidence of a successful Y2K audit and remediation of business-critical systems. That action by auditors made Y2K a Board-level business issue that was given high priority and major resources.

FAILURES AVOIDED

Thousands of errors were found and corrected during the 1990s, avoiding failures that would otherwise have occurred. Here are a few of them:

- The UK's Rapier anti-aircraft missile system[iv] The UK Ministry of Defence admitted that the Millennium Bug would potentially have left Britain's armed forces vulnerable to air attack because the anti-aircraft missiles might have failed to respond. The problem was identified inside the field equipment which activated the missiles and it could have made the system inoperable.
- Swedish nuclear plant. In mid-1998, a nuclear utility in Sweden decided to see what would happen if it switched the clocks in its reactor's computers to read January 1, 2000. The response surprised utility officials who had expected business as usual. The reactor's computers could not recognise the date (1/1/00) and shut down the reactor[v].
- An error was found and corrected in BP Exploration's oil platforms that would have caused significant disruption. BP Exploration told consultants that finding that one error justified their whole Y2K programmes[vi].
- US Coast Guards found[vii] errors that would have caused loss of steering control on ships, and failure of fire detection and control systems.
- Around 10% of VISA swipe-card machines were found not to handle cards that expired after 1999. Visa asked member banks to stop issuing cards with 00 expiry dates as a temporary measure[viii]. Visa then prepared to fine banks up to £10,000 per month until their systems became Y2K compliant[ix].
- Many thousands of date problems were found in commercial data processing systems and corrected. The task was huge: to handle the work just for General Motors in Europe, Deloitte had to hire an aircraft hangar and local hotels to house the army of consultants, and to buy hundreds of PCs.
- Many Y2K failures occurred in the 1990s and were then corrected. A typical example was an insurer who sent out renewals offering insurance cover with dates that ran from 1996 to 1900 rather than to 2000.

FAILURES AFTER 2000

Many failures occurred but were fixed and not widely reported[x]. They ranged from the trivial, to the serious; the trivial included the failure of the Runway Visual Range systems on all UK NATS airfields at 4am on January 1st (fixed by restarting the computers) and the serious included the failure of 15 nuclear reactors (in Spain, Ukraine, Japan and the USA)[xi].

One serious UK problem[xii] was only recognised when a health visitor in Yorkshire noticed an unusual number of babies that were born with Down's Syndrome. It transpired that more than 150 pregnant women were given the wrong results from pathology laboratory tests because the PathLAN computer system that was used in nine hospitals calculated the women's dates of birth incorrectly from January 2000: it had worked perfectly for the previous decade. The result was that women who should have been identified as being in a high-risk category were wrongly told that they did not need further testing.

EXAGGERATED FEARS?

There had been fears that there would be far worse problems than actually occurred. With hindsight, there were several reasons for this:

- A large number of problems had been found and corrected before 2000 as the result of the enormous international investment in Y2K remediation projects and by fixing the individual failures that had occurred over the preceding decade. The total cost was estimated at between US$300bn and US$500bn.
- The companies and countries that seemed to have started their Y2K programmes too late were rescued by the progress that was made by others e.g. by getting suppliers to correct software and equipment and in developing high-productivity tools for remediating legacy software. In 1990, remediation costs were several dollars per line of code; by 1999, this had fallen to fractions of a cent.
- The feared domino effect of cascade failures in interconnected systems did not happen because the major supply chains contained the largest companies and these companies had given high priority to finding and correcting problems in their systems and interfaces.
- Systems were actually not as tightly interdependent as had been feared so there was redundancy in supply chains that provided resilience.
- The threat had been exaggerated to some extent: partly by suppliers and consultants who wanted to sell equipment, systems and services; partly as a necessary step to attract attention to the seriousness of the threat and to stimulate and accelerate the required remediation action and contingency plans; partly by headline seekers; and partly by religious and survivalist sects that saw the new millennium as having apocalyptic significance.

LESSONS LEARNED?

The Y2K problem is often referred to now in support of an argument that expert warnings should be ignored; the threat from climate change comes to mind. This is completely the wrong conclusion to draw from the fact that the worst predictions did not occur.

The real lessons should be these:

- The problem was caused by poor software engineering. Systematic use of concepts such as abstraction, encapsulation, information hiding, and object-orientation could have allowed the construction of efficient programs in which the representation of dates could be changed easily when needed.
- Y2K was a particularly serious threat because it could cause many unrelated systems to fail simultaneously. Such shared-points-of-failure should be strenuously avoided.
- Systems were not as tightly coupled as had been feared; looser coupling and some redundancy provides useful resilience by reducing or removing the risk of cascading failures.
- It is possible to achieve remarkable urgency, effort and co-ordination once it is seen to be essential, and this can be very effective.
- Changes to audit standards and regulations cause Board-level attention and lead to appropriate action.

Y2K should be regarded as a signal event – a near miss that should serve as a major stimulus to change the way that IT systems are designed and built so that no similar problem can ever happen again. To an extent this happened. More companies appointed a CIO/CTO who instituted better management of IT systems, though only very rarely through good software engineering practices.

Y2K remediation should be seen as a major success showing that it is possible to deliver major IT system resilience on time given clear objectives, clear timescales, senior leadership attention, commitment to provide the necessary resources, clear communications and acceptance throughout the potentially affected organisations that the challenge is real and deserves the highest priority.

Unfortunately, little seems to have changed permanently perhaps because Y2K remediation was so successful and because it is only after a catastrophe has occurred that major reorganisations are brought about in the way an industry operates.

Almost all software-based systems are still designed and developed with cost and time-to- market taking priority over modularity, robustness, security, ease of modification and other soft- ware engineering principles. Testing is still the primary way in which programmers assure that software is fit for purpose, despite decades of evidence that testing can never find all errors and that mathematically rigorous software engineering methods and tools are cost-effective[xiii].

Shared points of failure are still introduced without considering the possible consequences. One current and critical example is the extraordinarily widespread dependence on the GPS signal for critical positioning, navigation and timing. Other examples are the almost universal reliance on encryption systems that could be destroyed by advances in quantum computing, and the growing dependence of multiple systems on identical software components or on identical sources of open data. The recent cyberattack on SolarWinds' Orion system is the latest illustration of the danger of software monocultures. WannaCry and NotPetya were earlier examples. There will be many more before the lesson is learnt.

In the interests of efficiency, supply chains have become far more tightly coupled and redundancy has been removed with little thought about the impact on resilience, making cascade failures more likely.

There is no government appetite for using regulation to encourage private companies to write better software. Yet, regulation has been used to improve the safety of children's toys and in many other product areas. Since the 1980s, industry has successfully argued that software is not a product but a service and that it should therefore be exempt from product safety and consumer protection laws.

The current cybersecurity crisis is one consequence of this failure to learn from Y2K but as there is no deadline to compel urgent action the risks to society from badly designed and insecure software are certain to continue to increase.

AUTHOR

Martyn Thomas is Emeritus Professor of IT at Gresham College and a Visiting Professor at the Universities of Manchester and Aberystwyth. He was awarded a CBE in 2007 for services to software engineering and has received honorary degrees from four universities.

REFERENCES

[i] https://www.gresham.ac.uk/lectures-and-events/what-really-happened-in-y2k

[ii] https://s3-eu-west-1.amazonaws.com/content.gresham.ac.uk/data/binary/2773/2017-04-04-MartynThomas_Y2K-T.pdf

[iii] Embedded systems, A guide to evaluating how this problem could affect your business, Millennium bug campaign brochure, February 1998, from Action 2000 [a UK Government awareness campaign]

[iv] The Times, 9 Dec 1998, p2

[v] Reported in http://xwalk.ca/Y2K.html

[vi] Reported to me by my Y2K consultants in Praxis/Deloitte Consulting, who had found the error.

[vii] Reported in http://xwalk.ca/Y2K.html

[viii] Reported in http://www.msnbc.com/news/42003.asp but no longer available online.

[ix] The Times (London, UK), Mon Jan 6 1997, Business News p41

[x] https://www.cs.swarthmore.edu/~eroberts/cs91/projects/y2k/Y2K_Errors.html

[xi] Originally on http://www.informationweek.com/wire/story/TWB20000102S0002 but removed

[xii] https://www.theguardian.com/society/2001/sep/14/NHS1

[xiii] https://www.gresham.ac.uk/lectures-and-events/making-software-correct-by-construction

www.nationalpreparednesscommission.uk

c/o Resilience First | 85 Tottenham Court Road, London, W1T 4TQ