CISQ

Consortium for Information & Software Quality ™

The Cost of Poor Software Quality in the US: A 2022 Report

From Problem to Solutions

HERB KRASNER
MEMBER, ADVISORY BOARD
CONSORTIUM FOR INFORMATION & SOFTWARE QUALITY (CISQ)
WWW.IT-CISQ.ORG
HKRASNER@UTEXAS.EDU
DATE: DECEMBER 15, 2022

Table Of Contents

List of Figures and Tables

Copyright Notice

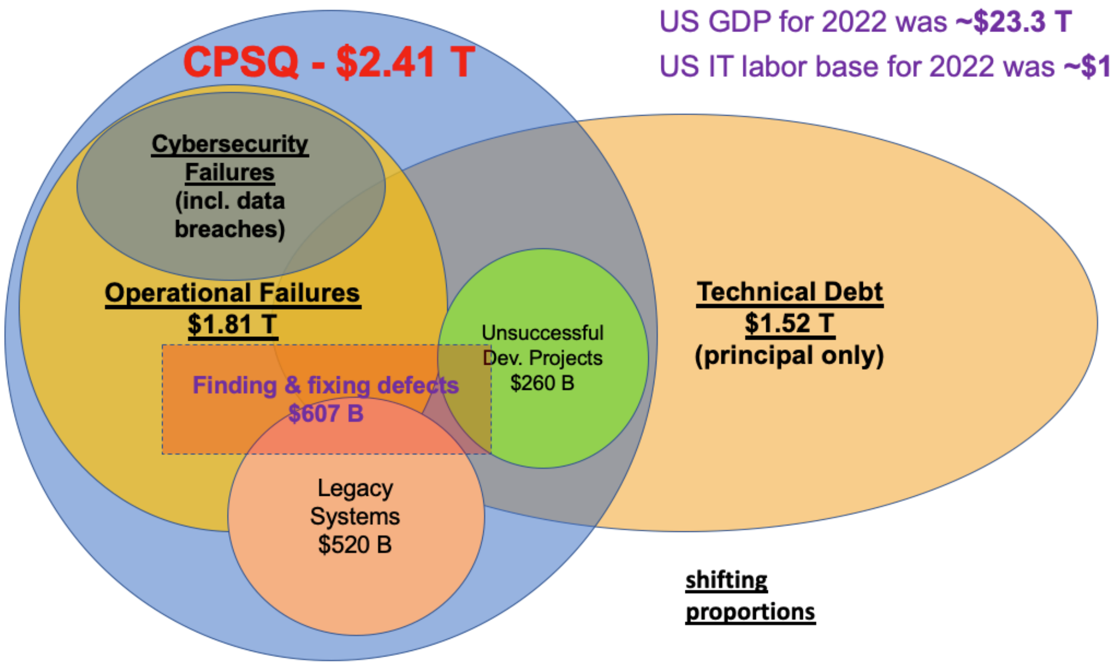
1. EXECUTIVE SUMMARY FOR 2022

The key US economic conditions that frame the context for this biennial report are:

- A projected GDP for 2022 of $23.35 trillion, a roughly 2% rise since 2020
- An inflation rate of 15% over the 2 year period
- A small 4% growth in the IT labor base over those 2 years to $1.51 trillion, and
- The number unfilled IT jobs sits at ~300,000 as of the end of August.

In this 2022 update report we estimate that the cost of poor software quality in the US has grown to at least $2.41 trillion[1], but not in similar proportions as seen in 2020. The accumulated software Technical Debt (TD) has grown to ~$1.52 trillion[1].

Figure 1-0 CPSQ in US in 2022


CPSQ - $2.41 T
US GDP for 2022 was ~$23.3 T
US IT labor base for 2022 was ~$1.51 T
Cybersecurity Failures (incl. data breaches)
Operational Failures $1.81 T
Unsuccessful Dev. Projects $260 B
Technical Debt $1.52 T (principal only)
Finding & fixing defects $607 B
Legacy Systems $520 B
shifting proportions

The 3 main problem areas that we will focus on this year are:

1. Cybercrime losses due to existing software vulnerabilities jumped way up
    - Losses rose 64% from 2020 to 2021. Those losses have not yet been determined for 2022.
    - Several critical infrastructure attacks cost an unmeasurable amount of pain and suffering over the last 2 years (e.g. Colonial Pipeline)
2. Software supply chain problems with underlying 3rd party components (especially Open Source Software, aka OSS) have risen significantly
    - In 2021, 77% of organizations reported an increase in the use of open source software
    - A medium-sized application (less than 1 million lines of code) carries 200 to 300 third-party components on average.

- The number of failures due to weaknesses in the open source parts of a software supply chain increased by 650% between 2020 and 2021.

3 The growing impact of Technical Debt (TD) has become the biggest obstacle to making any changes to existing code bases

- TD principle increased to ~$1.52 trillion (because deficiencies are not getting fixed).
- In spite of a projected rate of 15% growth in computer/IT positions created over the next decade, the number unfilled US IT jobs sat at about 300,000 at the end of August
- In late 2019 it was predicted that by 2025,40 % of IT budgets will be spent simply maintaining TD, and it’s a primary reason that many modernization projects fail.
- The number of weekly hours an average developer at a company spends on addressing “TD” is 13.5 out of 41.1, or 33% of their time.

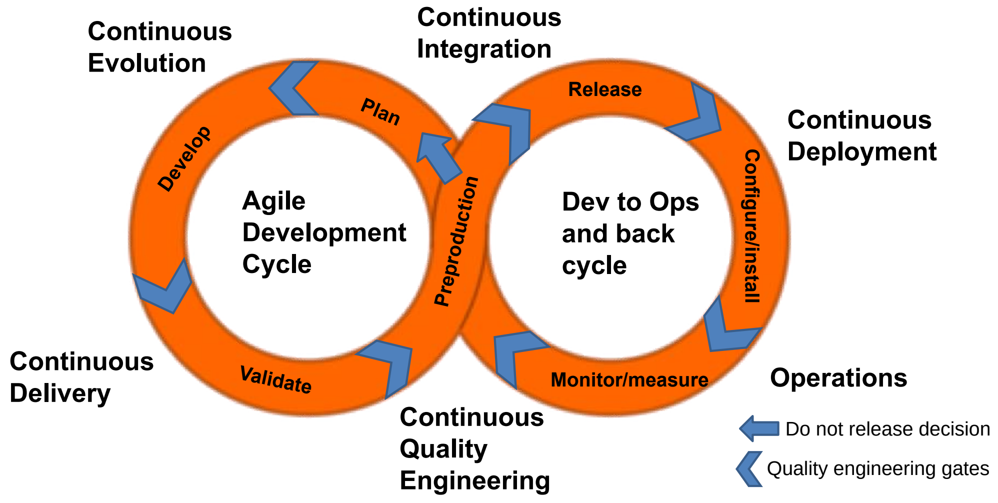
In this 2022 report we turn our attention to recent developments and emerging solutions to help improve the poor software quality situation as it now exists, and stabilize/reduce the growth rate of CPSQ in the near future.

The three main solution areas that we will focus on involve the emerging trends in modern tools for helping to find and fix software deficiencies, and standards and tools that can assist in identifying opportunities for reducing the growing TD. We see these as the most likely ways to start to get control over the poor software quality problem. We will focus on the emergence of:

- Quality standards/software problem taxonomies
- Tools for understanding, finding and fixing deficiencies/TD
- AI/Machine Learning (ML) tools for software engineering

All of these emerging solution areas can be focused around supporting the DevQualOps model that we introduced in 2020. Since software security is a subcategory of software quality, DevSecOps is therefore seen as a sub model of DevQualOps.

Figure 1-1 DevQualOps Model



Although the CPSQ and TD have risen significantly over the series of our three reports (the problem), so have the developments in the technology/practices to remediate those problems (solutions).

IT IS POSSIBLE THAT THE TREND IN OVERALL CPSQ WILL FLATTEN OVER THE NEXT DECADE IF ORGANIZATIONS WILL ADOPT THE RECOMMENDATIONS THAT WE HAVE PUT FORWARD IN THIS SERIES OF REPORTS. We hope that the solutions suggested herein become more widely adopted into the mainstream of software conception, development, production and evolution.

In addition to the broad recommendations of our previous reports, we add the following more specific recommendations for software development and IT organizations:

- Use the software quality standards, related measurements and tools that are emerging
- Analyze and assess the quality of all 3rd party/OSS components to be included in any system. Monitor them closely in operation. Apply patches in a timely fashion.
- Avoid DevOps and CI/CD models that do not include continuous quality engineering best practices and tools.
- Integrate continuous TD remediation into your SDLC
- Invest in the professionalism and knowledge of your software engineers.
- Consider having your developers certified for knowledge of the critical code and architectural weaknesses in ISO/IEC 5055 when OMG makes its "Dependable Developer' certification test available in late 2023 or 2024.

Our next report is tentatively planned for 2024, when hopefully some of the solutions identified in this report will catch up with the problems and show up in a positive change to the CPSQ trend.

Footnote 1 – See Appendix B for the detailed cost estimation methodology used

2. SUMMARY OF THE PREVIOUS CPSQ REPORTS

This is the third report in our biannual series on the cost of poor quality software. In our first two reports on this subject we focused on identifying the problem areas in software quality that could be viewed through the lens of additional costs due to poor quality. In doing so those reports helped to spread the news that this was indeed a major problem worthy of solutions that might be brought to bear. In this report however we start to outline more specific solutions to the underlying problems identified in our first two reports.

2018 report

In our 2018 report we focused on defining software quality and the categories of poor software quality that would allow us to better pinpoint the problem areas and symptoms of the poor quality issue. Our objective was to create a first-order estimate of CPSQ in the US by examining known information in several reported categories that we identified from a broad search of references. These main categories identified were: 1) legacy system problems, 2) losses from software failures, 3) troubled/cancelled projects, 4) finding and fixing defects, and 5) software TD. That report laid basic concepts and definitions which we referred to in the updates. Those definitions and concepts were:

- Common abbreviations used: IT, US, LOC, CoSQ, CPSQ (pg. 3)
- What is software (pg. 6)
- How much was being spent on IT/software at that time (pg. 7)
- The iceberg model of hidden software quality costs (pg. 10)
- What are legacy software maintenance costs (pg. 12)
- Summary of the major software failure stories in the news (pg. 16)
- What is software TD (pg. 19)
- The impact of available talent on software quality and its costs (pg. 21)
- What is software quality (pg. 28)
- The definition of the cost of software quality model (pg. 30)
- Our conclusions about the total CPSQ in analyzed categories (pg. 36)

These definitions and concepts remain valid except that in the intervening four years the definition of software quality has become more standardized and thus more measurable.

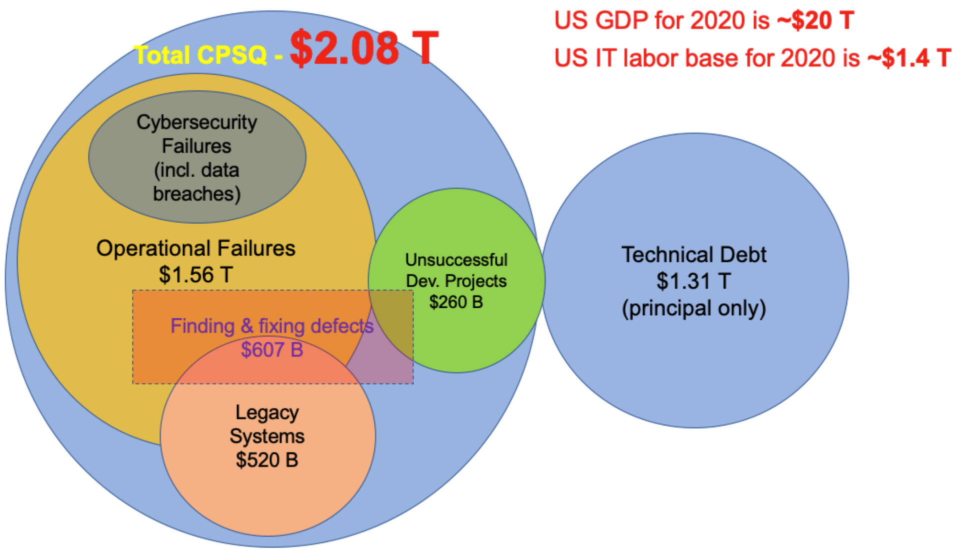
2020 report

In our 2020 report, which gained much more attention, we elaborated many of the publicly known failure reports to emphasize the sheer magnitude of the poor software quality problem. We laid out most of the strategies, tactics, models and best processes and practices that might be used to tackle the problem via a coherent approach that organizations could use. We concluded that report by describing the DevQualOps model that could be implemented by organizations and projects for which high quality was a goal. This strategy and model also were used to present our recommendations for all levels of an organization, starting with the C-suite all the way down to the software engineers and related disciplines.

In 2020 we identified what specific actions you can take at the level of: 1) individual software professional, 2) team/project leader, and 3) management/executive level of an organization. We also revealed an important (but little known) IBM study that explains the difference in practices between high performing vs. low performing software organizations. That study revealed a 5-10X difference in performance between the top 10% and the bottom 10% of organizations sampled. When you dig deeper into the data, the reason is clearly the adoption of quality management best practices.

In our 2020 report, which gained much more attention, we estimated that the cost of poor software quality in the US that year was $2.08 trillion, broken down as seen in the figure below:

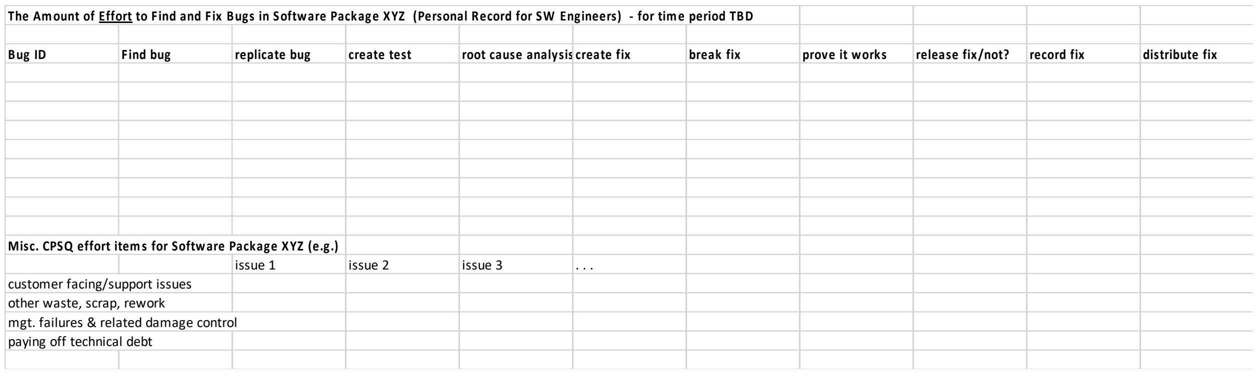
Figure 2-1 CPSQ in 2020


Total CPSQ - $2.08 T
US GDP for 2020 is ~$20 T
US IT labor base for 2020 is ~$1.4 T
Cybersecurity Failures (incl. data breaches)
Operational Failures $1.56 T
Unsuccessful Dev. Projects $260 B
Finding & fixing defects $607 B
Legacy Systems $520 B
Technical Debt $1.31 T (principal only)

We elaborated many of the publicly known failures to emphasize the sheer magnitude of the poor software quality problem. We laid out most of the strategies, tactics, models and best processes/practices that might be used to tackle the problem via a coherent approach that organizations could use. We concluded that report by describing the DevQualOps model that could be implemented by organizations and projects for which high quality was a goal. This strategy and model were used to present our recommendations for all levels of an organization, starting with the C-suite all the way down to the software engineers and related disciplines.

We recommended in 2020 that capturing the essential data necessary to determine your own cost of poor software quality was possible using today's tools. Quality oriented organizations are in fact capturing the internal effort data necessary to determine their organizational cost of finding and fixing deficiencies in the problem reporting and bug tracking systems that they now use. In some cases where such tools are not available, a simple spreadsheet suffices. This example form can be filled out in 1 minute at the end of each day. What is often missing in lower maturity organizations is the will to do so.

Figure 2-2 Software Engineer's CPSQ effort spreadsheet



The Amount of <u>Effort</u> to Find and Fix Bugs in Software Package XYZ (Personal Record for SW Engineers) - for time period TBD										
Bug ID	**Find bug**	**replicate bug**	**create test**	**root cause analysis**	**create fix**	**break fix**	**prove it works**	**release fix/not?**	**record fix**	**distribute fix**
Misc. CPSQ effort items for Software Package XYZ (e.g.)										
		issue 1	issue 2	issue 3	. . .					
customer facing/support issues										
other waste, scrap, rework										
mgt. failures & related damage control										
paying off technical debt										

See Appendix A for a more detailed summary of the 2020 Report.

The remainder of this report dives deeper into the most pressing software quality problems that are emerging in practice and related technologies, and to recent developments and emerging solutions to help improve the poor software quality situation as it now exists. We start by looking at a sample of the biggest operational software failure stories of the last 2 years.

3. SOFTWARE FAILURE STORIES AND EMERGING DEFICIENCY MODELS

Since our 2020 report, the state of software quality in the US has not gotten any better and appears to be getting worse. This can be seen in the following selected subset of software failures stories over the last two years. There are undoubtedly many more accounts that have not received this level of public attention.

The year 2021 started off with a bang as the extent of the SolarWinds hack became known and then through the year as the actual costs and impacts of that hack were revealed. In the middle of the 2-year period, the Colonial Pipeline attack showed us the vulnerabilities in our critical infrastructure, and then the Log4j hack showed us that widely used open systems technology was vulnerable. As we close in on the end of 2022, several new failure stories have emerged (e.g. Wintermute, GIT, BBN Chain) to remind us of the expanding nature and impact of these software failures.

Table 3-1 - Biggest Software Failures Of 2021- 2022

Headline	Explanation/Impact	Description
SolarWinds Orion:2020-21 (aka "sunburst hack") In the first nine months of 2021, the Orion breach cost SolarWinds $40 million, and soon escalated to $90 million, which then included incident response and forensic services for companies who were impacted by this incident and have cyber insurance coverage. By summer 2021, we learned that the SolarWinds attack cost affected companies an average of $12 million. Companies in the U.S. reported an average of a 14% impact on their annual revenue. SolarWinds stock plummeted from	SolarWinds Orion is an enterprise network management software suite that includes performance and application monitoring and network configuration management along with several different types of analyzing tools. It is common for network administrators to configure SolarWinds Orion with pervasive privileges, making it a vulnerable target. According to reports, the malware introduced (presumably by Russia's Foreign Intelligence Service) affected many companies and organizations. Even government departments such as Homeland Security, State, Commerce and Treasury were affected, as there was evidence that emails were missing from their systems. A supply chain compromise of a Dynamic Link Library (DLL) was found at the heart of the vulnerability. SolarWinds reported that just over 18,000 of their clients downloaded an affected version, though not all were actively hacked. Approximately 100 known companies were impacted.	On December 13, 2020, CISA released Emergency Directive 21-01: Mitigate SolarWinds Orion Code Compromise, For more specifics see this advisory. The perpetrators then proceeded to add malicious code into one of the company's most used software services, Orion. The hacking incident was stealthy and nondestructive, allowing it to slip under SolarWinds' radar and stay there for months. The code spread itself to other clients by hitching a ride on one of the regular software updates that SolarWinds sends out to its clients. There, the malicious code set up a backdoor for the hackers, allowing them to install even more invasive malware and spy on their targets and leak any information they deemed important.

Headline	Explanation/Impact	Description
$25/share at the end of 2020 to a current price of about $9/share.		
T-Mobile data breach affects 50 million customers On March 18, 2021, a bad actor illegally accessed and acquired personal data of more than 50 million customers. T-Mobile learned about the massive data breach on August 17, 2021.	The type of personal information that had been compromised varied by individual but included names, dates of birth, addresses, phone numbers, drivers' licenses, government identification numbers, social security numbers, and T-Mobile prepaid PINs. To protect their customers, the mobile carrier informed them of the security breach and encouraged them to take proactive steps regularly to keep their data safe. Though T-Mobile CEO apologized for the data security breach and promised to beef up defenses, many customers affected by the data breach have decided to take legal action. The lawsuits allege that T-Mobile's poor security protocols are to blame and allowed hackers to gain access to the company's services and extract the personal information of millions of people. The total cost to T-Mobile and its customers is yet to be determined.	A vulnerable router was used to gain access to T-Mobile's servers. T-Mobile's security was described by the hacker as awful.
TikTok glitch resets followers to zero On May 3, 2021 when TikTok users logged on to the app the last thing they expected to see was all of their users gone.	TikTok experienced a glitch that displayed the wrong followers/following count. Some users even had trouble accessing the app, with the app blocking their accounts. Users took their frustration to social media. Soon #TikTokDown was trending. More and more users came forward asking the social media giant to fix the glitch and restore their accounts and followers. TikTok confirmed the glitch, letting their users know that they were working on repairing the issue. The glitch was resolved overnight,	A company with such a large user base cannot let software bugs slip through. Glitches like this can easily be prevented with better software testing.
Colonial Pipeline's costly ransomware attack The attack on Colonial Pipeline is one of the worst cyber-attacks	Colonial Pipeline is the largest refined products pipeline in the U.S., a 5,500 mile (8,851 km) system involved in transporting over 100 million gallons from the Texas city of Houston to New York Harbor. It carries 45% of the fuel consumed on the U.S. East Coast. On April 29, hackers gained access to Colonial Pipeline's network through a virtual private	The ransomware attack on Colonial Pipeline shows the extensive damage that insufficient security measures and system vulnerabilities can cause. The attack began when a hacker group identified as DarkSide accessed the Colonial Pipeline network. The attackers stole 100 gigabytes of data within a

Headline	Explanation/Impact	Description
that occurred in 2021-22. This attack disrupted nearly half of the fuel supply on the East Coast of the United States. It caused gasoline shortages in the Southeast and a spike in fuel prices.	network (VPN) account, which allowed employees to remotely access the company's network. The hackers obtained valid credentials that enabled them to breach Colonial Pipeline's network, because the VPN account did not use multifactor authentication. After a week, on May 7, Colonial Pipeline received a ransom note demanding a cryptocurrency ransom be paid. Shortly after, the pipeline was shut down. Delivering roughly 2.5 million barrels of fuel across the Southeastern United States daily, the outage crippled fuel delivery. It resulted in long lines at gas stations—some of which ran out—and higher fuel prices. The hackers stole nearly 100 gigabytes of data and threatened to leak it if they didn't pay the ransom. Colonial Pipeline paid a ransom of 75 Bitcoins ($5 million) to the hackers, who were believed to be the cybercrime group known as DarkSide. The total cost of the underlying security vulnerabilities is incalculable.	two-hour window. Following the data theft, the attackers infected the Colonial Pipeline IT network with ransomware that affected many computer systems, including billing and accounting. Colonial Pipeline had to shut down the pipeline systems to prevent the spread of the ransomware. Once they paid the DarkSide hackers to get the decryption key, they were able to restart their systems. The root cause appears to be a stolen password to Colonial's VPN.
Twitter A vulnerability in Twitter's software exposed an undetermined number of owners of anonymous accounts to potential identity compromise over the last year. It was reported that data on 5.4 million users were offered for sale online.	Data obtained from the exploitation of that vulnerability was being sold on a popular hacking forum for $30,000. The vulnerability allowed someone to determine during log-in whether a particular phone number or email address was tied to an existing Twitter account, thereby revealing the account owners. A security researcher discovered the flaw in January, informed Twitter and was paid a reported $5,000 bounty. The bug was introduced in a June 2021 software update and was fixed.	The revelation of the breach came while Twitter was in a legal battle with Tesla CEO Elon Musk over his attempt to back out from his previous offer to buy San Francisco-based Twitter for $44 billion. The breach is especially worrisome because many Twitter account owners, including human rights activists, do not disclose their identities in their profiles for security reasons that include fear of persecution by repressive authorities.
Facebook ranking bug A group of Facebook engineers identified a "massive ranking failure" that exposed as much as half of all	The engineers first noticed the issue in October 2021, when a sudden surge of misinformation began flowing through the News Feed. Facebook's Downranking system failed to properly demote probable nudity, violence, and even Russian state media. The issue was	The technical vulnerability was apparently introduced in 2019 but didn't create a noticeable impact until October 2021. In a large complex system like this, bugs are inevitable and understandable. What happens when a powerful social

Headline	Explanation/Impact	Description
News Feed views to potential "integrity risks"	internally designated a level-one SEV — a label reserved for high-priority technical crises. Unable to find the root cause, the software engineers watched the surge subside a few weeks later and then flare up repeatedly until the ranking issue was finally fixed on March 11, 2022. The damage to Facebook's reputation for moderating controversial content was incalculable.	media platform has one of these faults? How would users even know?
Tesla recalls almost 12,000 vehicles In November, 2021 Tesla recalled close to 12,000 vehicles after discovering a glitch in its Full-Self Driving beta software.	Following its most recent update on October 23, Tesla began receiving reports from customers reporting that their vehicles had falsely identified forward collision threats which caused the automatic emergency braking (AEB) system to activate and bring the vehicle to a sudden stop, raising the risk of a rear end collision and injury to those within the vehicle. Tesla discovered a communication error in the 10.3 Full-Self Driving (FSD) beta software. Namely, the software bug indicated a false forward collision	To mitigate potential security risks, Tesla asked its quality assurance team to investigate and identify the cause of the software bug. The automaker promptly released a Safety Recall Report to recall affected vehicles—certain Model S, Model X and Model 3 vehicles manufactured 2017-2021, and certain Model Y models manufactured 2020-2021. Tesla released a separate update to address the software issue and notified vehicle owners of the issue and resolution. Thankfully, there were no crashes or injuries as a result of the software bug.
Grand Theft Auto What promised to be a high-quality remaster of the Grand Theft Auto classics—GTA III, Vice City and San Andreas—turned out to be a low-quality game full of bugs, glitches, and poor design decisions.	When the game was released in November 2021, the reception from fans was far from great, and Rockstar Games received a lot of backlash. Some users even went as far as to ask for a refund. Why? because the quality was bad—really, really bad. The NPC graphics were terrible, the character models were blotchy, the frame rate constantly dropped, the rain effects made it difficult to see, missions and minigames did not work as intended, and the audio quality was appalling. All these issues together made the game almost unplayable. The cost to the company's reputation has not been calculated.	The video game publisher has since uncovered—and apparently fixed—the long list of software bugs. Nevertheless, the damage was done, and it will take a long time for the publisher to recover from this blunder.
Log4j software bug leaves millions of web servers vulnerable	What makes this bug so terrifying is the fact that Log4j, an open-source logging library, is used by many companies worldwide, including high profile organizations like Apple, Amazon, Cisco, IBM, Microsoft, and many more. Many parties—	Efforts are being made to fix the issue (called log4shell). Teams around the world are working hard to patch affected systems before hackers can exploit them, while organizations are

Headline	**Explanation/Impact**	**Description**
The Log4j software bug set the internet on fire after it left millions of web servers vulnerable to hackers. The vulnerability was first discovered in the beginning of December, 2021, and its impacts have carried over into 2022 and are still being felt today.	companies, clients, and users alike—are very worried. The Log4j software is used to record all activities happening in a wide range of systems, such as errors and routine system operations, and deliver diagnostic messages to system administrators and users. The most common example of Log4j at work is the 404 error message that everyone is familiar with. Hackers can exploit these diagnostics to scan for vulnerable systems to install malware, steal credentials, and gain confidential data. Due to the extent of damage it could potentially cause, many believe that the Log4j software bug is the worst vulnerability in years. Hackers are using it to trick victims into mining small amounts of cryptocurrency for them and to hack private Minecraft servers. It's a combination of a new vulnerability being simultaneously widespread and easy to exploit. The Netherlands National Cyber Security Centre has identified hundreds of common software applications that are vulnerable to the flaw if not updated, and a number that may be not have a patch yet available. In a blog post, Microsoft said it has observed China, Iran, North Korea and Turkey exploiting it.	urged to install the latest security updates in order to counter the threat as soon as possible. While sweeping through their networks and applying a patch might be a solution for now, many companies are still left vulnerable and this solution may still not be enough. Only time will tell. Log4j is the most prominent incident which has contributed to last year's 650% year-on-year increase in OSS supply chain-targeted attacks.
Wintermute, Sept. 2022 Hackers stole digital assets worth around $160 million from crypto trading firm Wintermute. The hack involved a series of unauthorized transactions that transferred USD Coin, Binance USD, Tether USD, Wrapped ETH,	According to a May 2022 report from Bishop, Fox security incidents pummeling DeFi platforms resulted in losses to the tune of **$1.8 billion i**n 2021 alone, with the services experiencing an average of five hacks per month. To make matters worse, malicious actors have stolen $1.3 billion worth of cryptocurrency in the first three months of 2022 alone, in comparison to $3.2 billion that was stolen for the entirety of 2021, indicating a "meteoric rise" in crypto crimes.	The attack targeted the decentralized finance (DeFi) space. The OSS package implicated is Profanity, an Ethereum vanity address generation tool, where recently it was disclosed that a vulnerability could be abused to recompute the private wallet keys from addresses created using the utility.

Headline	**Explanation/Impact**	**Description**
and 66 other cryptocurrencies to the attacker's wallet.		
Spyder Python IDE, Sept. 2022 (OSS) Originally disclosed in August 2007, the bug has to do with how a specially crafted target archive can be leveraged to overwrite arbitrary files on a target machine simply upon opening the file.	As many as 350,000 open source projects are believed to be vulnerable to exploitation as a result of a security flaw in a Python module that has remained unpatched for 15 years. The open source repositories span a number of industry verticals, such as software development, artificial intelligence/machine learning, web development, media, security, and IT management.	Now tracked as CVE-2007-4559 (CVSS score: 6.8), it is rooted in the tarfile module, successful exploitation of which could lead to code execution from an arbitrary file write.
BNB Chain, Aug. 2022 BNB Chain, is a blockchain linked to the Binance cryptocurrency exchange. BNB, which stands for 'Build and Build' (formerly called Binance Coin), is the blockchain gas token that 'fuels' transactions on BNB Chain, as noted earlier this year.	In August it was reported that an estimated $2 billion worth of cryptocurrency had been stolen in 13 cross-chain bridge attacks, accounting for 69% of total crypto funds stolen in 2022 so far.	This is the latest in a series of major incidents targeting cross-chain bridges – which facilitate transfer of assets between blockchains – this year, after those of Axie Infinity, Harmony Horizon Bridge, and Nomad Bridge.
GIT, August, 2022 Git is a hugely popular open-source version control system, counting more than 80 million active users.	In August, a vulnerability in the open source development tool Git which, if not addressed, allows bad actors the keys to the kingdom. In total, 332,000 websites were found as potentially vulnerable, including 2,500 residing on the .gov domain. GitHub users are being targeted with malicious copies of legitimate repositories. While the majority of malicious code changes were made in the last couple of months, with some found to be dating back seven years. As reported by GitHub, a threat actor managed to steal data from "dozens of victims".	OSS technology has always had the potential for flaws, being rooted in publicly accessible code. This type of vulnerability, on such a popular platform, can have "serious consequences" for affected firms. More recently the RepoJacking bug was discovered that could allow an attacker to take control over a GitHub repository, and potentially infect all applications and other code relying on it with malware.

Headline	Explanation/Impact	Description
OpenLightSpeed, November 10, 2022	OpenLiteSpeed is the open source edition of LiteSpeed Web Server, the sixth most popular web server, accounting for 1.9 million unique servers across the world. Palo Alto Networks Unit 42 said that multiple high-severity flaws have been uncovered in the OSS code that could be weaponized to achieve remote code execution.	The vulnerabilities discovered include: 1. Remote Code Execution (CVE-2022-0073) rated High severity (CVSS 8.8) 2. Privilege Escalation (CVE-2022-0074) rated High severity (CVSS 8.8) 3. Directory Traversal (CVE-2022-0072) rated Medium severity (CVSS 5.8)
FTX Crypto Hack, Nov. 11, 2022 **FTX, a $32 billion company, vaporized overnight.**	On November 11,2022 the CEO of crypto exchange firm FTX resigned and said that they were filing for bankruptcy. Apparently $473 million in crypto assets were stolen. FTX was "investigating abnormalities" regarding movements in crypto wallets "related to consolidation of FTX balances across exchanges." The stablecoins and other missing tokens were being quickly converted to Ether, the second-largest cryptocurrency after Bitcoin, on decentralized exchanges, which is a common technique used by hackers to prevent their funds from being seized.	The underlying facts are still unclear at this time.
CommonSpirit Health, Oct.-Nov. 2022	A crippling ransomware attack on the second-largest U.S. nonprofit health system is showing what happens when critical health care infrastructure goes down. The attack on CommonSpirit Health, which has 142 hospitals in 21 states, left IT locked, delayed surgeries and caused widespread disruptions in patient care. It also left millions of patients waiting at least two weeks to learn if their personal information was compromised.	The underlying facts are still unclear at this time.

As we go to press, even more are showing up: e.g.

Healthcare ransomware attacks, Sept. 2022
- https://thehackernews.com/2022/10/cisa-warns-of-daixin-team-hackers.html?_m=3n%2e009a%2e2869%2eye0ao43m8z%2e1u6n

Text4Shell – October, 2022
- https://thehackernews.com/2022/10/hackers-started-exploiting-critical.html?_m=3n%2e009a%2e2868%2eye0ao43m8z%2e1u63

OpenSSL high severity vulnerabilities – patch for 2 now available – Nov. 1, 2022
- https://thehackernews.com/2022/11/just-in-openssl-releases-patch-for-2.html

Critical Vulnerabilities in 3 Industrial Control Systems Software – Nov. 8, 2022
- https://www.cisa.gov/uscert/ncas/current-activity/2022/11/03/cisa-releases-three-industrial-control-systems-advisories

The Top Software Weaknesses Of 2021-2022 Identified

All of the software failure stories presented in the previous table are due to weaknesses found in the software of those systems. Software weaknesses are: flaws, bugs, vulnerabilities, or various other types of deficiencies found a software solutions' code, architecture, implementation, or design. They potentially expose the systems containing that software to failures and/or cyber-attacks.

MITRE recently shared their top 25 most common and dangerous weaknesses impacting software over the previous two calendar years. These are considered the most dangerous because they're usually easy to discover, come with a high impact, and are prevalent in software released during the last two years.

To create this list, MITRE scored each known weakness from its CWE database (Common Weaknesses Enumeration) based on its prevalence and severity after analyzing the data for 37,899 CVEs (Common Vulnerabilities Enumeration) from NIST's National Vulnerability Database (NVD) and CISA's Known Exploited Vulnerabilities (KEV) Catalog.

Although these weaknesses have been applied primarily to the quality aspect of security, we assert that they more generally apply to software quality. The table below provides insight into the top 25 most critical and current weaknesses.

Table 3-2 Top 25 CWEs

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

Many professionals who deal with software will find the CWE Top 25 a practical and convenient resource to help them mitigate their software quality risk.

In April, 2022 in partnership with the FBI and the NSA, cybersecurity authorities worldwide published their list of the top 15 most exploited security flaws, with links to the National Vulnerability Database entries and associated malware.

Table 3-3 Top 15 CVEs

CVE	Vulnerability	Vendor and Product Found In	Type
CVE-2021-44228	Log4Shell	Apache Log4j	Remote code execution (RCE)
CVE-2021-40539		Zoho ManageEngine AD SelfService Plus	RCE
CVE-2021-34523	ProxyShell	Microsoft Exchange Server	Elevation of privilege
CVE-2021-34473	ProxyShell	Microsoft Exchange Server	RCE
CVE-2021-31207	ProxyShell	Microsoft Exchange Server	Security feature bypass
CVE-2021-27065	ProxyLogon	Microsoft Exchange Server	RCE
CVE-2021-26858	ProxyLogon	Microsoft Exchange Server	RCE

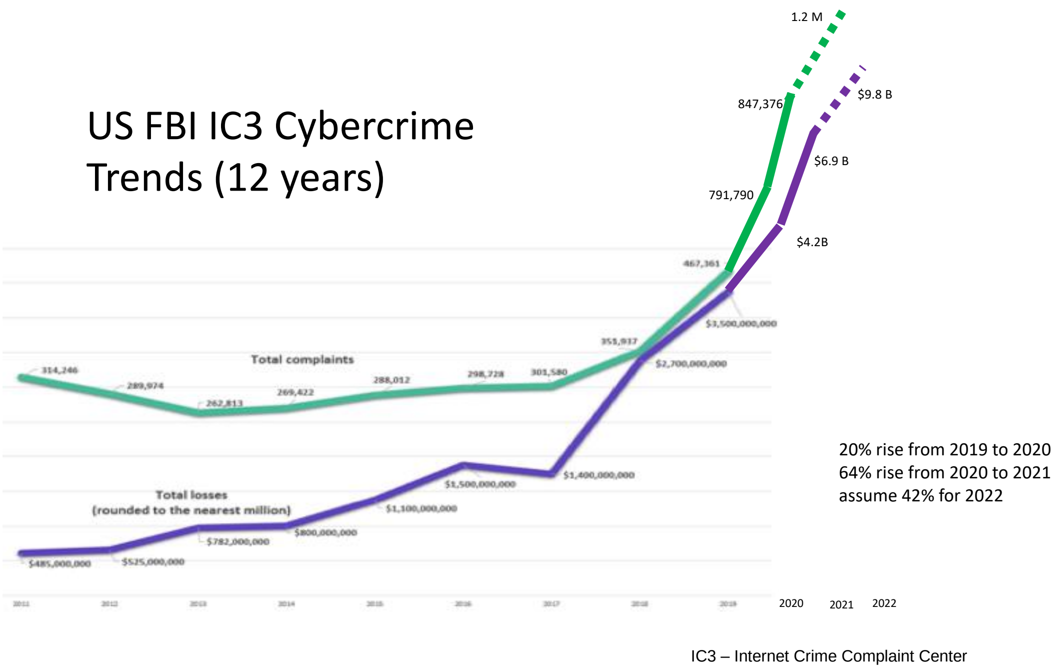
CVE	Vulnerability	Vendor and Product Found In	Type
CVE-2021-26857	ProxyLogon	Microsoft Exchange Server	RCE
CVE-2021-26855	ProxyLogon	Microsoft Exchange Server	RCE
CVE-2021-26084		Atlassian Confluence Server and Data Center	Arbitrary code execution
CVE-2021-21972		VMware vSphere Client	RCE
CVE-2020-1472	ZeroLogon	Microsoft Netlogon Remote Protocol (MS-NRPC)	Elevation of privilege
CVE-2020-0688		Microsoft Exchange Server	RCE
CVE-2019-11510		Pulse Secure Pulse Connect Secure	Arbitrary file reading
CVE-2018-13379		Fortinet FortiOS and FortiProxy	Path traversal

The related mitigation measures that should help decrease the risk associated with these were published. When the above databases are linked to the emerging standards for software quality, and their underlying patterns, then the industry has a solid basis for measuring and controlling software quality.

4. THE RISING COST OF CYBERCRIME

Cybercrime is predicted to cost the world $7 trillion USD in 2022, according to Cybersecurity Ventures. If it were measured as a country, then cybercrime would be the world's third largest economy after the U.S. and China. The number of cybercrime incidents and their related costs have been on the rise for over a decade, as seen in the chart below. This data is based only on those incidents that have been reported to the FBI, which of course, is much less than the total picture of all US cybercrimes.

Figure 4-1 Cybercrime Trends in the US: Last 12 years



The total costs associated with cyber-attacks -- lawsuits, insurance rate hikes, criminal investigations and bad press -- can put a company out of business quickly.

Headlines from the cybersecurity industry

Plenty of cybersecurity news broke in 2021-22. Hackers and cybercriminals ruthlessly attacked businesses, governments and individuals. Here's a look at some of the major industry headlines:

- According to VMware's "The State of Incident Response 2021" report, 82% of surveyed organizations are concerned their company is vulnerable to a cyber-attack. The report found that 49% of organizations lack the expertise and tools for adequate incident response.
- The FBI's Cyber's Most Wanted list features more than 70 individuals and groups that have conspired to commit the most damaging crimes against the U.S. These crimes include computer intrusions, wire fraud, identity theft, espionage, theft of trade secrets and many other offenses.
- VPNs are especially vulnerable, since Six Chinese companies own 30% of VPNs, and 97 of the top VPNs are run by 23 parent companies, many of which are based in countries with lax privacy laws.

- Organizations are conducting more application security testing scans than ever before, according to the Veracode "State of Software Security v12" report. In 2021, most firms were scanning applications approximately three times a week -- up from three times a year in 2010.
- Security attacks increased 31% from 2020 to 2021, according to Accenture's "State of Cybersecurity Resilience 2021" report. The number of attacks per company increased from 206 to 270 year over year.
- According to Debricked, on average it takes *over 800 days* to discover a security flaw in OSS. For instance, the Log4shell (CVE-2021-44228) vulnerability was undiscovered for 2649 days.

Similar reports have been published as well:

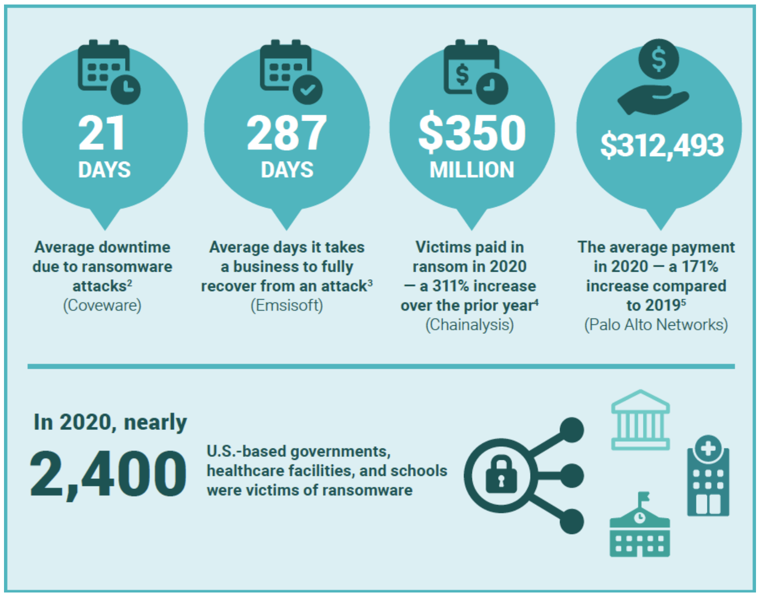
1. Cybercriminals can penetrate 93 percent of company networks (betanews.com)
2. Software supply chain attacks hit three out of five companies in 2021 | CSO Online
3. 82 percent of CIOs believe their software supply chains are vulnerable (betanews.com)
4. Businesses Suffered 50% More Cyberattack Attempts per Week in 2021 (darkreading.com)
5. Ransomware attacks, and ransom payments, are rampant among critical infrastructure organizations - Help Net Security
6. Ransomware Trends, Statistics and Facts in 2022 (techtarget.com)

The hottest cybercrime trends in 21-22 were:

1. Ransomware
2. Cryptojacking
3. Deepfakes
4. Videoconferencing attacks
5. IoT and OT attacks
6. Supply chain/OSS attacks
7. Extended Detection and Response solutions (aka XDR)
8. Critical infrastructure attacks

The financial impact of Ransomware is best seen in the following summary chart from the April, 2021 IST Ransomware Task Force Report.

Figure 4-2 – Ransomware Impact in 2020



More recently, the Cybersecurity and Infrastructure Security Agency reported in February 2022 that it is aware of ransomware incidents against 14 of the 16 U.S. critical infrastructure sectors.

Perhaps no cybersecurity trend was bigger in 2021-22 than the scourge of supply chain ransomware attacks.

- Among the biggest attacks was the Colonial Pipeline ransomware attack, which affected the East Coast of the U.S. in May 2021.
- There were ongoing issues related to supply chain security stemming from a breach at software management vendor SolarWinds.

As of 2022, the average cost of a data breach in the United States amounted to $9.44 million, up from $9.05 million in the previous year.

Another good way to see how quickly cybercrime has become a major problem is by the amount of money that various organizations will pour into that area. For example:

- Last year, Google committed $10 billion over 5 years to fund a program to strengthen cybersecurity, including expanding zero-trust programs, helping secure the software supply chain, and enhancing open-source security.
- The Biden administration requested $2.1 billion in the 2022 discretionary budget for the Cybersecurity and Infrastructure Security Agency (CISA). That is an increase of $110 million from the 2021 level and builds on the $650 million provided for CISA in the American Rescue Plan. The money would go to:
  1. Enhancing its cybersecurity tools
  2. Hiring experts
  3. Obtaining support services to protect and defend federal technology systems
  4. Creating a Cyber Response and Recovery Fund

5. SOFTWARE SUPPLY CHAINS (SSC) WITH OSS

A software supply chain is composed of the components, libraries, tools, data and processes used to develop, build, publish and evolve a software system. Software builders often create their products in large part by assembling open source, third party and commercial software components. This way of building software allows rapid feature development and massive reuse of existing code but opens the doors to supply chain vulnerabilities. Open source software (OSS) plays a critical role in today's IT ecosystem. The overwhelming majority of modern codebases contain open source components, with open source often comprising 70% or more of the overall code. According to CAST Software, a medium-sized application (less than 1 million lines of code) carries 200 to 300 third-party components on average.

The top 4 reasons cited for using OSS are:

- Access to innovations and latest technologies
- No license cost, overall cost reduction
- To modernize technology stack
- Functionality to improve development velocity

In 2021, according to Perforce, 77% of organizations reported an increase in the use of open source software, with 36% indicating a significant increase. Only 1.6% of over two thousand respondents indicated that they reduced the usage of open source software. Yet, only 13% are concerned that their OSS is unsecured or untested, whereas 27.5% have no reservations in using OSS. There would appear to be a disconnect between the risks these organizations are taking relative to the actual risks of using certain OSS components.

According to a recent IDC survey report, 86% of respondents said they sometimes or always try to find open source options over other kinds of software. However, most organizations are unaware of the extent to which they already use open source and underestimate their dependency on it, a dependency that comes with some risks. IDC research found that 68% of organizations that use any kind of open source software acknowledged they had been impacted by a vulnerability or compromise associated with an open source technology over the past two years. Threats from open source vulnerabilities impact net-new applications that enterprises are building, critical legacy applications, and software offered by suppliers.

The mass reuse of open-source components and libraries has dramatically accelerated the development cycle and the ability to deliver functionality according to customer expectations. But the counterpart to this gain has been a loss of control over the origin of the code that goes into a production system. This chain of dependencies exposes organizations and their customers to vulnerabilities introduced by changes that are outside of their direct control.

The number of attacks using the open source ecosystem as a propagation vector reaching software supply chains increased by 650% between 2020 and 2021. The European Cybersecurity Agency (ENISA) predicted that supply chain attacks will increase fourfold by 2022. Other experts have predicted even higher.

These pre-existing vulnerabilities can enable an attack that targets the less-reliable components of a system's supply chain. This can trigger a failure that creates a chain reaction that propagates to a network of providers and then spreads through the Internet to many other interconnected systems. These attacks are targeting the source code of the components of these software systems.

The best example in the last two years was the SolarWinds attack, where a flawed software update, hiding a devastating virus, reached up to 18,000 customers, including high-profile companies and government institutions worldwide. In the case of software security—there has been a 430% increase in SSC attacks. In the recent SolarWinds attack, a simple customer software update delivery included a devastating virus. This infected update reached 425 of the Fortune 500 companies, which included telecommunication companies, accounting firms, government, and academic institutions.

This was soon followed by Log4Shell, which exploited a vulnerability within the Apache Log4j logging utility. The large impact in this case lies in the widespread use of this Java library and the possibility for more attackers to have remote code loaded and executed by the logger. There is huge potential damage (i.e.costs), as happened in the Log4j, SolarWinds, Mimecast, Ledger, Kaseya, Ethereum and SITA.

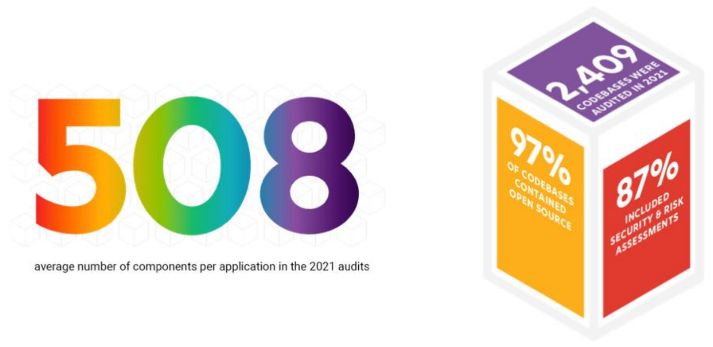
Relevant Studies Reveal The Extent Of This Problem

The Synopsys Black Duck Audit database represents open source activity from over 20,000 sources worldwide. Their 2020 report (the 5th of their series) described their 2019 study of the audit findings from 1,253 commercial codebases in 17 industries. By codebase they mean the source code and libraries that underlie an application, service, or library. Their 2020 findings included the following:

- 82% of the open source components found were out of date (i.e., unpatched or not well supported)
- 99% of codebases audited contained open source components
- Open source made up 70% of the audited codebases (doubled in 5 years)
- 75% of codebases contained vulnerabilities (up from 60% in 2018), and 49% contained high risk vulnerabilities (e.g. Heartbleed)
- An average of 82 vulnerabilities were identified per codebase
- The most frequent languages found were: JavaScript (74%), C++ (57%), shell (54%), C (50%), Python (46%), Java (40%), TypeScript (36%), C# (36%), Perl (30%), Ruby (25%)
- The top 10 open source components found (in order of occurrences) were: jQuery, Bootstrap, Font Awesome, Lodash, jQuery UI, Underscore-stay, Inherits, isArray, Visionmedia and Minimatch

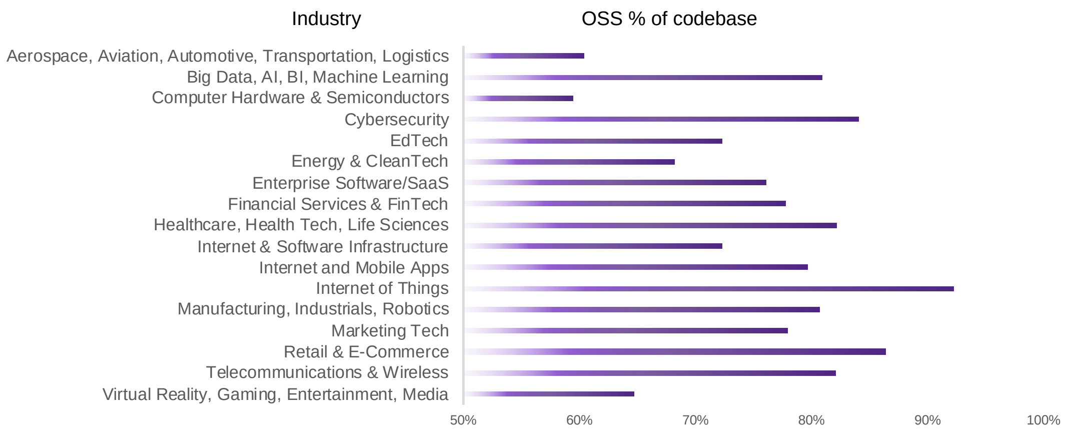
There is more recent data on the extent of the OSS problem from their 2021 study. Their 2022 report (the 6th of their series) described their 2021 study of the audit findings from 2,409 commercial codebases in 17 industries. They showed that OSS remains ubiquitous and pervasive.

Figure 5-1 Synopsys OSS component survey results


508
average number of components per application in the 2021 audits
2,409 codebases were audited in 2021
97% of codebases contained open source
87% included security & risk assessments

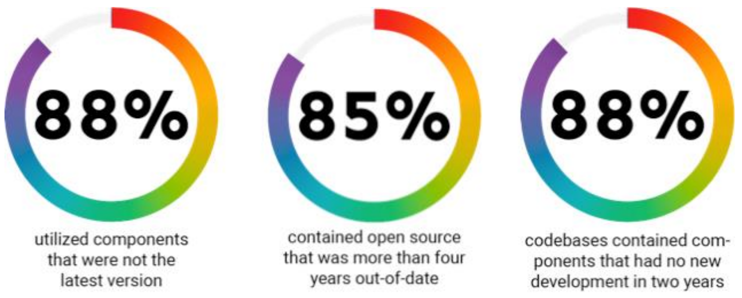
When they examined the percentage of the code bases (X-axis below) which was OSS by industry, they were able to show the following:

Figure 5-2 Synopsys OSS component survey results – by industry



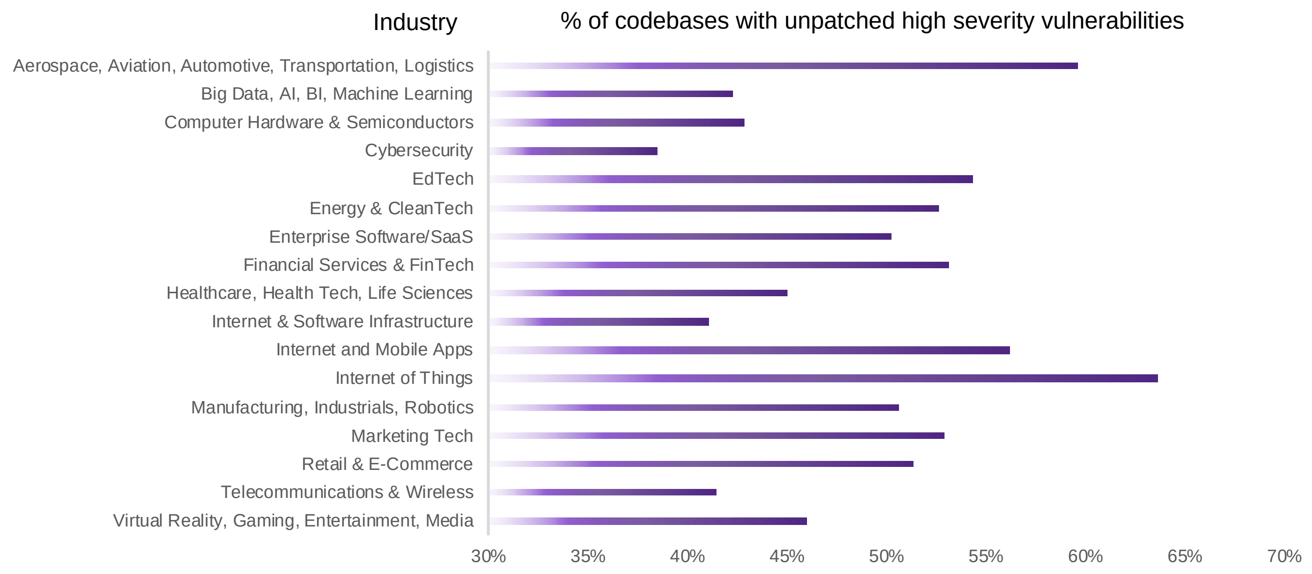
They were able to show the extent to which organizations are still struggling to track and manage their OSS.

Figure 5-3 Synopsys OSS component survey results – not maintained


88%
utilized components that were not the latest version
85%
contained open source that was more than four years out-of-date
88%
codebases contained components that had no new development in two years

They were able to show what percentage of the codebases contained unpatched high severity vulnerabilities by industry.

Figure 5-4 Synopsys OSS component survey results – unpatched high severity bugs



Another Synopsys survey report explored the strategies that organizations around the world are using to address open source vulnerability management as well as the growing problem of outdated or abandoned open source components in commercial code.

A report based on Snyk customer scans from Jan. 1–Sept. 30 of this year (skewed in favor of Java ecosystems) found that the top 10 most prevalent critical and high vulnerabilities in OSS were:

1. Denial of Service (DoS)
2. Remote Code Execution (RCE)
3. Deserialization of Untrusted Data
4. SQL Injection
5. Prototype Pollution
6. Insecure Temporary File
7. Directory/Path Traversal
8. Privilege Escalation
9. Regular Expression Denial of Service (ReDoS)
10. NULL Pointer Dereference

Another useful source of information is the NIST's National Vulnerability Database (NVD), which lists the known vulnerabilities of the major commercial software vendors, such as: Oracle, Microsoft, IBM and Adobe Those four account for nearly 17% of total vulnerabilities, for all products and versions combined.

In 2019 an analysis of the NVD was performed and the following results of vulnerabilities by vendor and by weakness types were published (charts provided by CAST.)

Figure 5-5 NIST vulnerability survey results – by vendor

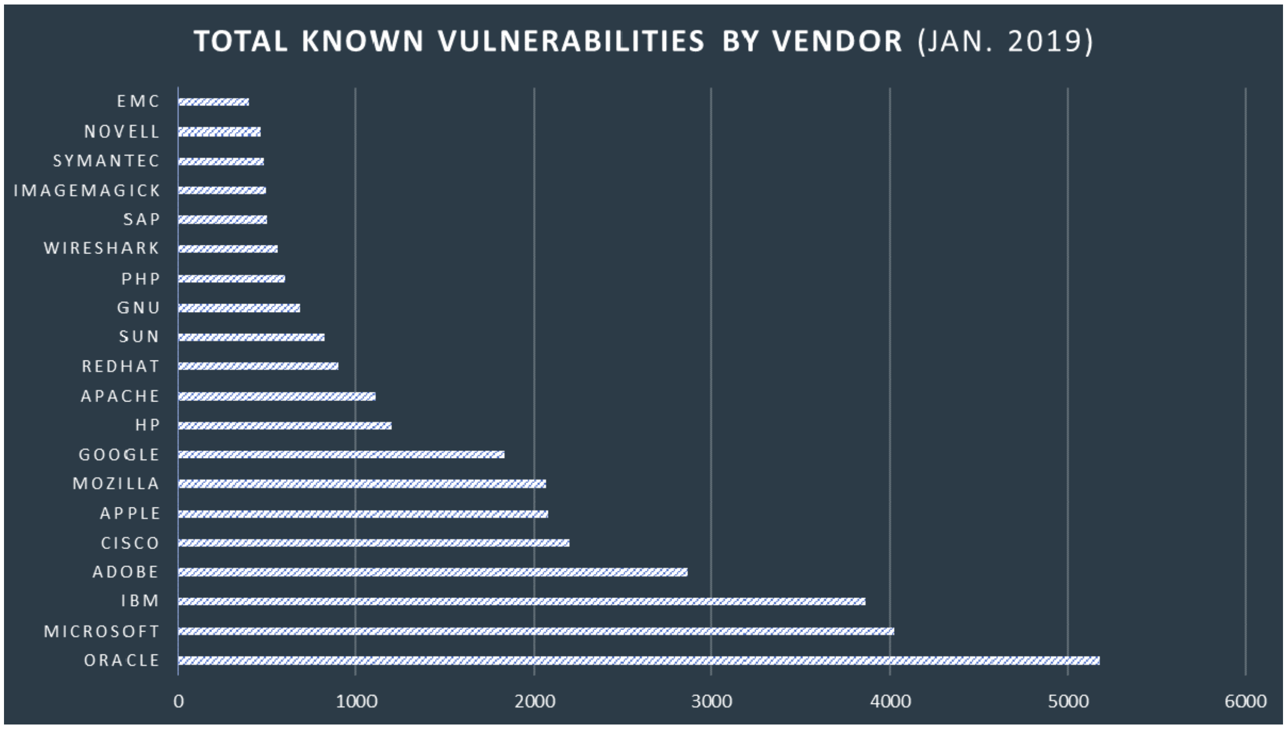
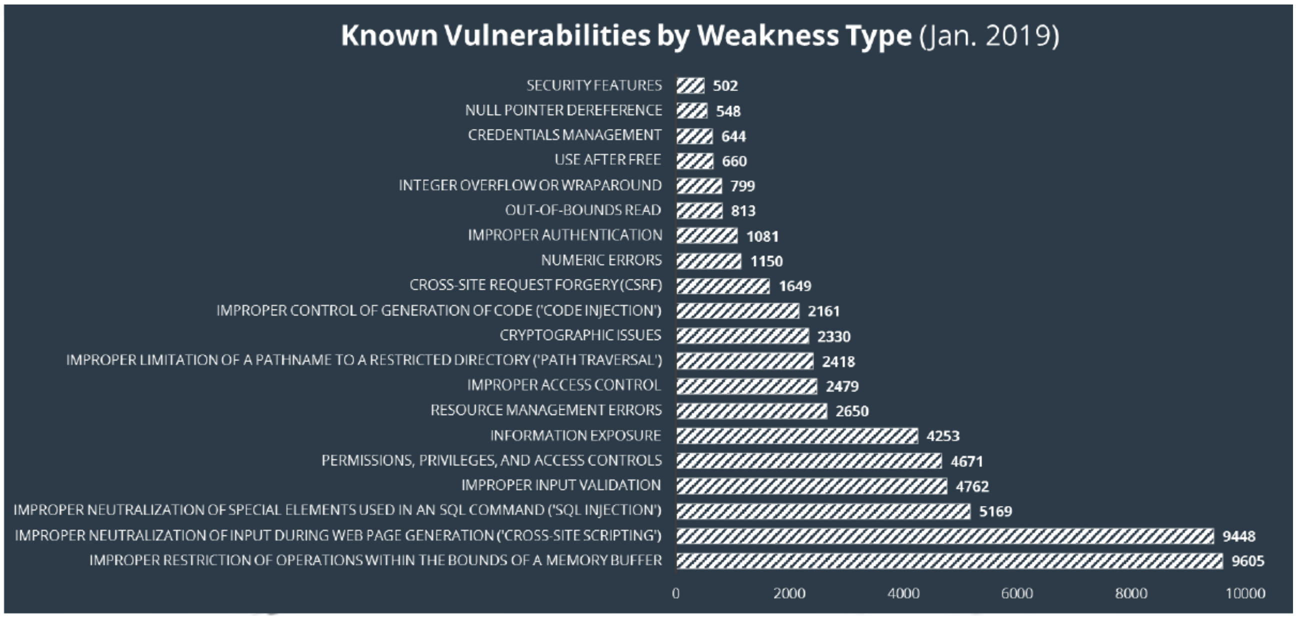


Figure 5-6 NIST vulnerability survey results – by weakness



A more recent similar report has not been published since 2019, but would be most valuable if it were done today.

OSS Best Practices Recommended

Currently popular open source software platforms include GitHub, Fat Free CRM, InfluxDB, D3.js, R, TensorFlow, Keras, Serverless, Apache Airflow, Activiti, PrestaShop, and OpenCart. The quality of these software components is empirically unknown, and therefore might potentially introduce flaws that compromise success.

Each programming language has unique structural flaws which might lead a developer into creating a flaw. For example, low-level languages like Assembly, C, or C++ are vulnerable to buffer overflow which hackers can exploit to write malicious code to adjacent memory once buffer capacity is full. Another common vulnerability found in languages like SQL, JavaScript, and PHP is code injection, where hackers exploit flaws in data processing that cause user input to be interpreted as system commands or include malicious script in uploaded files. A good resource for identifying language specific problems is the software bug framework report, which includes a taxonomic hierarchy of weaknesses that applies to all languages. It contains specific suggestions to avoid weaknesses that arise from constructs that are incompletely specified, exhibit undefined behavior, are implementation-dependent, or are difficult to use correctly.

Some practical advice for organizations include:

- Manage the software supply chain like you manage any other critical corporate risk.
- The next Log4J-like vulnerability is inevitable. You need to be ready. Form a software incident response team (SIRT) to protect your software supply chain.
- Don't forget about COTS. Many independent software vendors (ISVs) use open source software liberally. You need to scan binaries and software build dependencies.
- It helps to know where your greatest exposure lies. Inventory everything to know what you have, and understand the composition of your applications and the pervasiveness of open source components across your application portfolio. Establish and maintain a software inventory or a Software Bill of Materials (aka, SBOM).
- Continuously analyze the software supply chain by integrating source code scans into your DevQualOps CI/CD pipelines. Rapidly mitigate known vulnerabilities to reduce the exposure time. Continually monitor software components against databases of known vulnerabilities.
- Build as much assurance for included code (i.e., open source software, libraries, and packages) as for the code that you natively develop.

6. TECHNICAL DEBT (TD)

TD accumulates when decision makers go for a short-term solution to a software development problem—instead of a more exhaustive, long-term solution—and this comes with substantial, initially hidden costs that organizations must pay later.

TD is also a measure of a company's burden that stems from aging and inflexible IT systems. In one McKinsey survey, 87% of global CIOs said the complexity of their existing systems prevents them from investing in the next generation of services. Global CIOs say that their total TD is between 20% and 40% of the total value of their "technology estate" before depreciation. Software AG says that 78% of organizations surveyed have accrued more TD in the past year than in previous years, but only 42% of companies feel that they have the ability to assess all of their TD.

Signs that an organization is overburdened with TD include:

- A backlog of project requests from business units.
- Contractors and consultants are being hired to fix or maintain existing systems.
- A rise in support cases about core functionality that is impaired.
- The IT department has decided not to upgrade software the company continues to use.
- When a relatively simple request for a modification turns into a major project.
- The amount of debt to be serviced limits the choices in using the IT budget
- The amount of unplanned work noticeably grows.

There are 2 parts to TD.

- *Principal* refers to the cost of refactoring/modifying software artifacts so that they reach a desired level of maintainability and evolvability.
- *Interest* is the extra effort that developers will spend when making those changes because of the existence of TD, which accumulates over time as software becomes more brittle. Every minute spent on not-quite-right code adds interest on the debt.

TD is the result of a suboptimal construct that is expedient in the short term, but sets up a technical context that can make a future change costlier or even impossible. Much of the current debt that exists today was created by "quick and dirty" development techniques (e.g. agile without software engineering discipline). In any case, all or part of this debt may need repayment. When or if to repay involves difficult tradeoffs.

There are many types of TD, such as requirements, architectural, code, testing, and operational. TD can be injected at any stage of software development, spreading across other phases and system parts and causing various problems. And just as we have seen in the CPSQ, preventing TD or removing it early, is the most cost effective long term strategy.

TD (like the CPSQ) is important because it helps facilitate the discourse between engineers and management on how to best invest limited resources on corrective maintenance and code improvement versus adding new features and functionality. The tipping point is reached, for example, when the cost of new features, bug fixes and maintenance exceed the project budget, causing them to reach a state of technical bankruptcy.

The Rough Cost Of TD In 2022

According to Stripe the number of hours an average developer at a company spends on addressing "TD" is 13.5 out of 41.1, or 33%. A 2019 Scandinavian study revealed that developers waste, on average, 23% of their time due technical debt.

In our 2020 report we estimated the TD principle in the US to be ~$1.3 trillion, which would increase to **$1.52 trillion in 2022** due to inflation alone. This figure is roughly equal to the total dollars spent on the entire US IT labor base in 2022. We have no good estimates yet on the accumulating interest. Nonetheless, we recognize that TD is a huge problem, which will get much worse if we do nothing at all. The potential of managing TD is seen in the Stepsize research which revealed that organizations who actively manage tech debt will ship at least 50% faster.

Progress In The Measurement Of Software TD

One of the main problems in dealing with TD has been the lack of a way to measure that debt. To help overcome that problem, CISQ/OMG led the development of an Automated TD (ATD) measurement standard, which is currently being updated with a new version expected in 2023.

The ATD standard estimates the effort to correct all instances of the software weaknesses included in the ISO/IEC 5055:2021 Automated Source Code Quality Measures standard that remain in a software application's code at release. This estimate can be used to predict future corrective maintenance costs. This measure is calculated by static analysis tools.

The cost to fix structural quality problems constitutes the principal of the debt, while the inefficiencies they cause, such as greater maintenance effort or excessive computing resources, represent interest on the debt. The measure expresses the cost of software quality in terms a business can understand by estimating future corrective maintenance costs to remedy structural defects in code.

CISQ surveyed developers in a number of organizations to estimate how long it would take to fix each of the weaknesses in well-constructed code. The estimates provided default values for the effort to fix each weakness. To calculate TD, we adjust the default value for each occurrence of a specific weakness by factors that affect the difficulty of fixing it such as the complexity of the component, its exposure to the rest of the system, etc. The adjusted efforts for each occurrence are summed to produce a total remediation effort for that weakness. The total remediation effort for the weaknesses in a quality characteristic are summed to create a remediation effort for that characteristic. Finally, the remediation efforts for the four quality characteristics (Reliability, Security, Performance Efficiency, and Maintainability) are summed to produce the TD measure.

Where Else To Look For Automated Solutions To Finding And Fixing TD

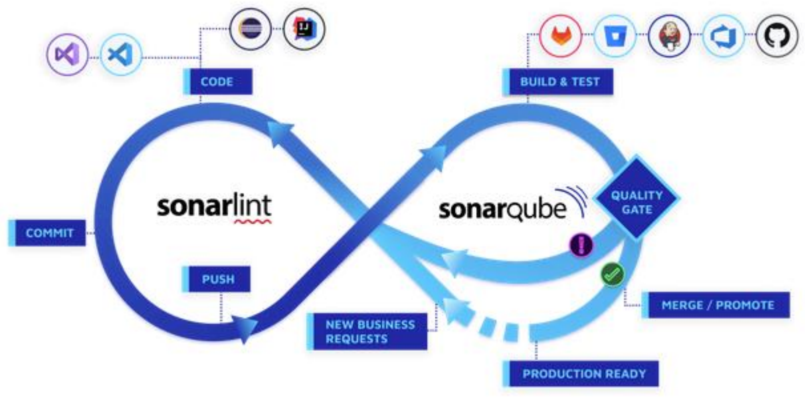
In the past decade, TD R&D advanced from a concept toward specific engineering practices and tools for identifying, monitoring, and remedying TD issues. We believe it will take several more years before TD management grows from adolescence to adulthood.

Nonetheless, some useful tools have come into existence as static code debt analyzers. For examples: SonarQube, CAST, Synopsys and NDepend. Standards are lacking across these tools for: TD metrics, indices, quality models, static analysis rules, TD remediation models, and definitions of the various TD concepts. Measuring TD will be necessary to estimate principal and interest to prioritize TD management in practice. Technologies such as CAST MRI detect not only weaknesses at the code unit level, but also weaknesses in the architecture such as layer-skipping calls.

Tools such as SonarQube/SonarLint, have developed addons to estimate a TD principal based on a "code smell" and rule violation model. The downside of these tools are that they create the impression that TD consists of low-level code deficiencies and nothing else - rather than the much costlier architecture and dispersed quality characteristics.

Many modern static analysis tools support our DevQualOps model, and also supports many languages and most popular configuration management tools.

Figure 6-1 Sonar DevOps model



In the next few years, we can expect to see more works investigating the impact TD has on internal qualities, such as faultiness, reliability, and code maintainability, maybe with increased support for nonobject- oriented languages. When making the argument for repaying TD, interest should include not only maintainability but other forms, such as operating expenses, opportunity costs, security, user experience problems, and product value.

The current advice for software engineers when dealing with TD is to:

- pay particular attention to how internal dependencies are created, as there is a fine balance between changeability and the number of dependencies per module: too many, and they become entangled, making the system hard to modify and too few, and the system is hard to modify because fewer modules are reused "as is" (a tree-like dependency graph), resulting in multiple modules implementing similar functionality (and applying the same change to all of them is duplicative).

- carefully balance how these dependencies are created by devising clear architectural rules that prevent the creation of undesired dependencies that end up generating bad smells. (e.g. Arcan)
- stay aware of new TD analysis tools and adopt and use accordingly for refactoring analysis.
- continuously refactor – i.e. makes the changes to the internal structure of your software to make it easier to understand and cheaper to modify in the future without changing its observable behavior.

In An Empirical Study of Refactoring Challenges and Benefits at Microsoft, developers reported the following refactoring gains:

- Improved maintainability (30%)
- Improved readability (43%)
- Fewer bugs (27%)
- Improved performance (12%)
- Reduction of code size (12%)
- Reduction of duplicate code (18%)
- Improved testability (12%)
- Improved extensibility & easier to add new feature (27%)
- Improved modularity (19%)
- Reduced time to market (5%)

7. SOFTWARE QUALITY STANDARDS

In our 2018 report we provided the following discussion about the definition of software quality.

"Quality" can mean different things to different people. The concept and vocabulary of quality is elusive. The meaning differs depending upon circumstances and perceptions. The dictionary definition of Quality (in general)

1. the standard of something as measured against other things of a similar kind; the degree of excellence of something.
2. a distinctive attribute or characteristic possessed by something.

The ISO 8402 standard defines quality as "the totality of features and characteristics of a product or service that bears on its ability to satisfy stated or implied needs [now]." Quality is a different concept when focusing on a tangible software product versus the perception of a quality service enabled by software. For instance, ISO/IEC 25010 defines a model of the quality characteristics of a software or system product, while ISO/IEC 25019 defines the quality-in-use characteristics experienced when using such products. The meaning of quality is thus time-based or situational.

Consumers now view quality as a fundamental measure of their total perception/experience with a product or service, as well as of the company, delivery and maintenance network that provides and supports it — a kind of unified "quality-value" metric.

While the above may suffice for general discussions, there exists a need for each project to have its own more specific definition. Software quality is therefore more precisely described as a combination of the following aspects:

1. Conformance to requirements
   - The requirements are clearly stated and the product must conform to it
   - Any deviation from the requirements is regarded as a defect
   - A good quality product contains fewer defects
2. Fitness for use/purpose
   - Fit to user expectations: meet user's needs
   - A good quality product provides better user satisfaction
3. Meeting standards
   - In many industries and organizations certain external and internal standards must be complied with
   - A good quality product conforms to required standards of quality (ISO/IEC 25010 & ISO/IEC 5055) and the process used to develop it (CMMI, SPICE).
4. Underlying aspects, which include
   - Structural quality (E.g. complexity)
   - Aesthetic quality (E.g. appearance, ease of use, etc.)
   -

Every application or business domain faces a specific set of software quality issues, and software quality must be defined accordingly. A definition fashioned from the above aspects and/or applicable standards should be created for your organization and for each project.

In 2018 we discussed the difference between good and poor software quality and concluded that ***If there was a simple measure for "good" software, we'd all be using it, and everyone would demand it.***

Historically several metrics have often been used as indicators, usually in combination. For example:

- Defect trend over time is often used to differentiate - good is a decreasing curve, poor is an increasing curve.
- Testing code coverage has been used as a surrogate – but doesn't speak to the quality of the tests themselves.
- Cyclomatic complexity, depth of inheritance, degree of class coupling, structural complexity, and a few other metrics, are indicators of sub-par code.
- The amount of effort that it takes to understand what a piece of code does is another good indicator.

What has changed since 2018 is the emergence of widely recognized standards to help us deal with the thorny problem of measuring software quality.

The Emergence Of Standards For Defining Software Quality

The most useful and common framework that is available to help each project more precisely define their software quality goals is now the ISO/IEC 25000 series.

The ISO/IEC 25000 series of standards, known as SQuaRE (System and Software Quality Requirements and Evaluation), contains a framework to evaluate software product quality. ISO/IEC 25010 defines a set of eight software quality characteristics, or system "-ilities," i.e. security, reliability, and maintainability. ISO/IEC 25023 describes how to apply the quality characteristics to measure software product quality.

However, the measures defined in 25023 largely measure quality at the behavioral level rather than at the level of specific quality problems found in the source code.

To fill that gap CISQ led the creation of ISO 5055 which defined source code level measures for four of the 8 quality characteristics — Reliability, Performance Efficiency, Security, and Maintainability . These are now being automated in the development of new code quality analysis tools. Measures for these four quality characteristics have now been adopted as international standards in ISO/IEC 5055:2021.

Each ISO 5055 code quality measure for Reliability, Performance Efficiency, Security, and Maintainability is comprised of a selected set of weaknesses (CWEs) from the Common Weakness Enumeration (CWE) taxonomy. The CWE is a good reference point for developers and tools and codifies over **800** known software weaknesses. Each CWE is a known code pattern that is a potential failure point found in many existing systems.

Development teams can use the above code quality standards to evaluate the structural quality of software ahead of each release, thus preventing dangerous flaws from being delivered into operational settings, where they will be orders of magnitude costlier to find and fix.

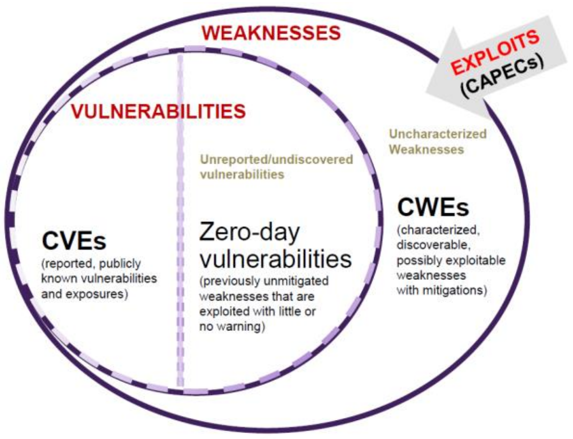
Although this is not the complete answer to ensuring high quality software production, it is a big step forward since our original report.

Many often consider "vulnerabilities" and "weaknesses" in software as interchangeable words. While they are related, they are different. These terms are defined in international standards. Standardized definitions for weaknesses and vulnerabilities are part of the ITU-T CYBEX 1500 series (CVE ITU-T X.1520, CWE ITU-T X.1524, CAPEC ITU-T X.1544) as outlined below.

- **Weakness:** Mistake or flaw condition in architecture, design, code, or process that if left unaddressed could under the proper conditions contribute to a cyber-enabled capability being vulnerable to exploitation; represents potential source vectors for zero-day exploits.
- **Vulnerability:** Mistake in software that can be directly used by a hacker to gain access to a system or network, or **Exposure:** Configuration issue or a mistake in logic that allows unauthorized access or exploitation.
- **Exploit:** Action that takes advantage of weakness(es) to achieve a negative technical impact.

The existence of an exploit designed to take advantage of a weakness (or multiple weaknesses) and achieve a negative technical impact is what makes a weakness a vulnerability. Weaknesses are listed in the Common Weakness Enumeration (CWE) Repository (cwe.mitre.org). Vulnerabilities (CVEs) are published in both the Common Vulnerabilities and Exposures dictionary (CVE.mitre.org) and the National Vulnerability Database (nvd.nist.gov). The methods bad actors use to exploit weaknesses and vulnerabilities are enumerated in the Common Attack Pattern Enumeration and Classification (capec.mitre.org). The figure below summarizes the relationships between these concepts.

Figure 7-1: Vulnerabilities, Weaknesses & Exploits


WEAKNESSES
EXPLOITS (CAPECs)
VULNERABILITIES
Uncharacterized Weaknesses
Unreported/undiscovered vulnerabilities
CWEs
(characterized, discoverable, possibly exploitable weaknesses with mitigations)
CVEs
(reported, publicly known vulnerabilities and exposures)
Zero-day vulnerabilities
(previously unmitigated weaknesses that are exploited with little or no warning)

Because the number, size, and complexity of software systems increases every day, so do the number of weaknesses and vulnerabilities. Cyber attackers use known vulnerabilities and weaknesses to exploit systems. Eliminating known vulnerabilities (CVEs) and the most egregious weaknesses (CWEs) would substantially reduce the impact from cyberattacks and data leakages. See the latest version of the CWE, the top 25 CWEs, and 138 CWEs in the ISO/IEC 5055 Automated Source Code Quality Measure standards developed by CISQ. If all new software (111 billion LOC per year globally) was created without these known vulnerabilities and exploitable weaknesses, the CPSQ would plummet.

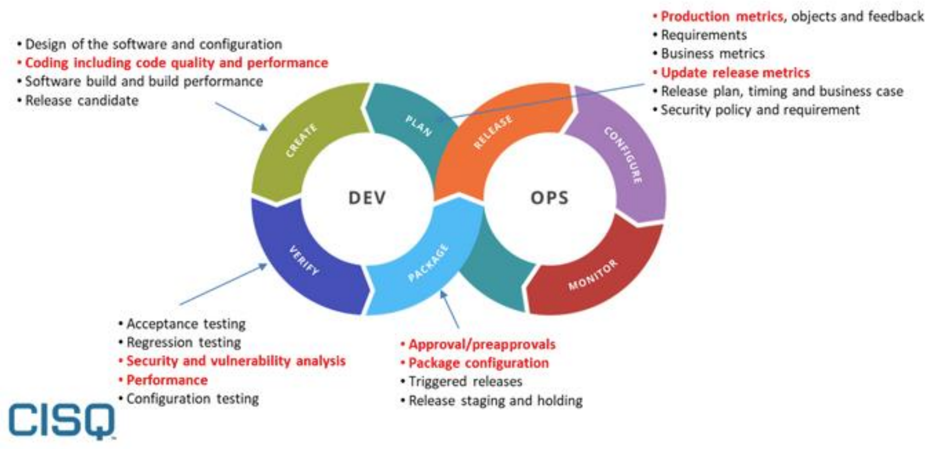
As indicated by the growth in data breaches, data protection and privacy are at the top of many organizational priorities. Many organizations will be undergoing process assessments associated with regulations to protect data, including General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), Health Insurance Portability and Accountability Act (HIPAA), and Cybersecurity Maturity Model Certification (CMMC).

Scanning code that will run or is running in enterprise network-connected assets that process or transmit data would determine if the systems or devices enable data leakage or lack adequate protections to mitigate unauthorized access to read or modify data. If so, then such a scan would reveal if the data protection/privacy controls associated with the process assessment were inadequately implemented.

To address this, CISQ developed an Automated Source Code Data Protection Measure (ASCDPM) that can be used in application security testing and software development to provide independent verification of processes revealing source vectors for data leakage or data corruption; providing indicators for non-compliance with respective data protection and privacy guidelines. Based on the CWE, the measure elements (weaknesses violating software quality rules) that compose the CISQ ASCDPM contain 36 parent weaknesses and 53 contributing weaknesses.

CISQ is directly addressing the DevQualOps model by their ongoing work, as shown in the figure below.

Figure 7-2 CISQ DevQualOps Model


• Design of the software and configuration
• Coding including code quality and performance
• Software build and build performance
• Release candidate
• Production metrics, objects and feedback
• Requirements
• Business metrics
• Update release metrics
• Release plan, timing and business case
• Security policy and requirement
CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR
• Acceptance testing
• Regression testing
• Security and vulnerability analysis
• Performance
• Configuration testing
• Approval/preapprovals
• Package configuration
• Triggered releases
• Release staging and holding
CISQ

8. UNDERSTANDING, FINDING AND FIXING DEFICIENCIES

As we were able to show in our 2020 report, all of the major categories of poor software quality costs are undergirded by the cost of finding and fixing the deficiencies that exist or are injected into software systems.

Ideally a software development shop should reduce bugs as much as possible before shipping, but in most situations, it becomes a tradeoff. To be competitive, an organization might want to deliver features or products to customers more quickly at minimum cost. The problem has always been that quality suffers in this tradeoff because it has been much harder to measure than both time and cost.

Most importantly, we know that bugs cannot be fully prevented: e.g. you can't test every single user scenario or all the execution paths in the code.

Empirical evidence suggests that organizations incorporating automated quality analysis and DevQualOps practices will observe improved quality through the improved discovery of deficiencies by integrating analysis as well as monitoring tools in their development and deployment environments.

Practically, a significant percentage of a software project's cost today is not spent in the creative activity of software construction but rather in the corrective activity of debugging and fixing errors. However, the task of debugging is inherently complicated. Most systems lack formal specifications describing intended program behavior. Without a formal or systematic documentation of correct behavior, the definition of an "error" or "bug" often resides in the software engineer's mind or in the user's sometimes nebulous expectations of program behavior.

Software development mistakes of all kinds—in source code, configurations, tests, or other artifacts—are a wide-ranging and expensive problem. Developers consume a significant proportion of engineering time and effort to understanding, finding and fixing bugs in their code, businesses lose market share when vulnerabilities in the software they sell impact customers, and overall productivity is impacted by software that does not work as intended or is prone to vulnerabilities.

The cost of finding and fixing deficiencies is the largest single expense element in the software development lifecycle. Over a 25-year life expectancy of a large software system, almost fifty cents out of every dollar will go to finding and fixing bugs. Large systems have much higher deficiency potentials that are more difficult to remove than for small systems due to size and complexity. The earlier in the development lifecycle deficiencies are found, the more economical the overall delivery will be. A good resource is the book, The Economics of Software Quality.

The CAST Crash 2020 report gave us insight into where to look for potential CPSQ improvements. Their latest benchmark on the structural quality of IT applications was developed from their database of 2,505 applications consisting of 1.549 BLOC (billions of lines of code), distributed across 533 organizations and 26 countries. The five software quality characteristics analyzed in their report are Robustness, Security, Performance Efficiency (aka Performance), Changeability, and Transferability. These are four out of the eight major software quality characteristics found in the ISO/IEC 25010 software product quality model standard (Changeability and Transferability are sub characteristics under Maintainability in ISO/IEC 25010).

CAST's findings were based on calculating the densities of critical weaknesses in applications and revealed that:

- The size of an application had negligible to no relation to its structural quality.
- Densities of critical weaknesses for Security were higher than those for Robustness and Changeability.
- The lowest densities were observed for Transferability (equivalent to Understandability).
- Industry segment is of lesser importance than other factors – but this only applies to Java-EE applications, for which Telecom, Software ISVs, and IT consulting had the highest densities of critical Robustness, Security, and Changeability weaknesses.
- Most industries showed wide variability in critical weakness densities and numerous extreme outlier scores.
- Security was the area where the mean densities of critical weaknesses and variability of scores were the highest.
- The factors that most affect quality attributes like Robustness, Security, or Changeability appear most likely to be specific to the application, the development team, and the specific conditions in the development environment.

Selected recommendations from the report were:

- Greater attention must be given to secure coding practices as many applications had densities of critical Security weaknesses that were unacceptably high. Security scores displayed wider variation than those of any other quality characteristic.
- Analyze source code on a regular basis prior to release to detect violations of quality rules that put operations or costs at risk. System-level violations are the most critical since they cost far more to fix and may take several release cycles to fully eliminate.
- Treat structural quality improvement as an iterative process pursued over numerous releases to achieve the optimal quality thresholds.

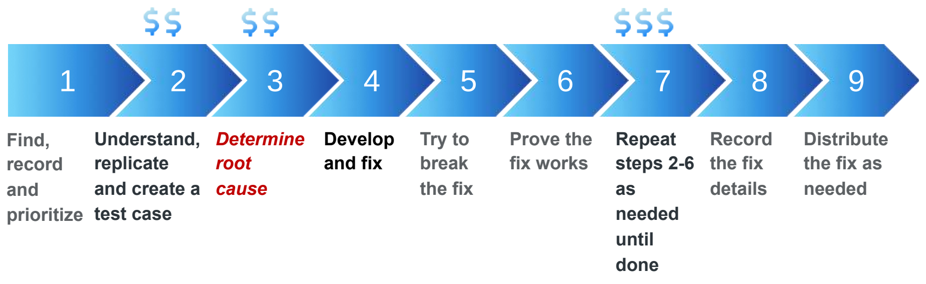
While adopting these evidence-based recommendations cannot guarantee high structural quality, they have been shown empirically to be associated with lower risk applications.

To better understand the costs involved here, we must look at the processes involved. First, since there is an order of magnitude cost difference between internal and external (deployed) deficiencies, we needed to understand those categories:

- Category 1 - Internal Deficiency Costs are costs associated with software deficiencies discovered before the system leaves the development organization and is deployed into the operational environment. These deficiencies occur when a system fails to meet a certain requirement (or critical coding/architectural rule), resulting in waste, scrap and/or rework. The deficiencies could be in the work products of development, the development process, and/or components if they fail to meet quality standards and requirements. Unfortunately, very few organizations track this category prior to the commencement of system testing.
- Category 2 - External Deficiency Costs are costs occurring when the failure of software to reach quality standards is not detected until after it is transferred into operation or to the customer. External failure/deficiency costs are incurred during customer use. The largest category of cost is professional effort to replicate, find, and fix all of the fielded deficiencies and re-appraisals to verify fixes.

Second, it is instructive to look at the software engineering process of understanding, finding and fixing deficiencies so that we may observe where the actual effort is expended. A simple model is shown in the figure below. The dollar signs indicate where most of the effort/cost is concentrated.

Figure 8-1: The Process of Understanding, Finding and Fixing Software Deficiencies



$$
$$
$$$
1
2
3
4
5
6
7
8
9
Find, record and prioritize
Understand, replicate and create a test case
Determine root cause
Develop and fix
Try to break the fix
Prove the fix works
Repeat steps 2-6 as needed until done
Record the fix details
Distribute the fix as needed

This process only grows in importance as software is continuously evolving and deployed and as society becomes increasingly dependent on software systems in all aspects of modern life. A high maturity version of this process then looks for other places in the code where the same mistake could have been made to correct any other occurrences, and then analyzes the root cause and eliminates it..

It helps to know where to look for deficiencies, vulnerabilities, weaknesses, bugs, refactoring opportunities, etc. Here are some practical tips:

1. At the code level certain languages are more prone to bugs than others (e.g. JavaScript – it's easy to learn, but it's also easy to inject bugs with it.)
2. At the design/architecture level, the more complex the components and interdependencies, the more likely it is to have complicated, confusing bugs.
3. Highly-interactive user interfaces, are most likely to have bugs.
4. Third-party libraries are more likely to have bugs. If it's open source, at least you can go and dig into the code. If its closed source you are often at the mercy of the vendor.
5. The first version of something new (beta) is guaranteed to have lots of bugs
6. Bugs that have escaped into the wild that represent unusual/unanticipated situations that depend upon usage context (e.g. corrupted user data) are some of the hardest to find

New Tools To Help

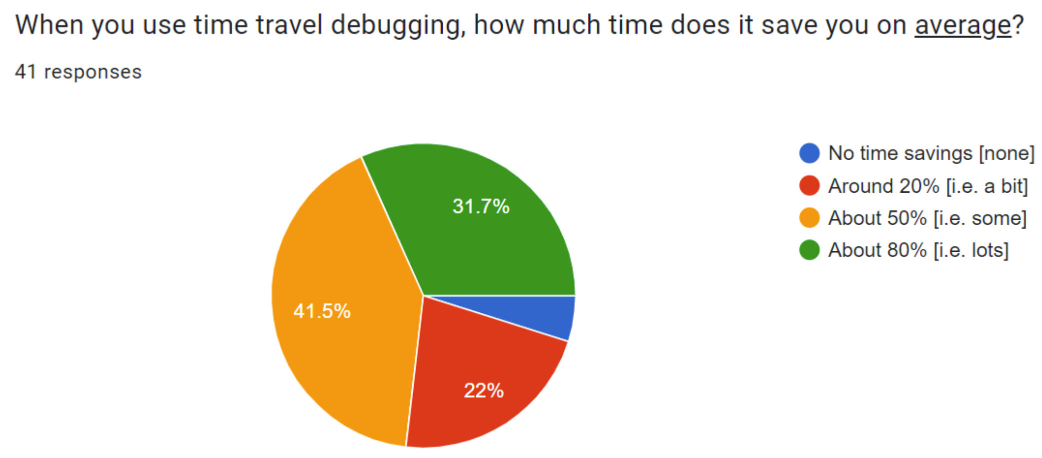
One of the new generation of debugging tools that has emerged to assist is called time travel debugging. Time travel debuggers allow software engineers to:

1. Backtrack in an application's execution history and inspect the complete state of the application at that point in time
2. Execute path navigation in both forward and reverse order, stepping, running, and using breakpoints, watchpoints, catchpoints, etc.
3. Make use of context-based navigation capabilities
4. Enable dynamic logging and reviewing
5. Replay from a failed state

6. Make use of annotation based collaboration capabilities

This ability to find and fix deficiencies more quickly is best demonstrated by the following study data. In November,2022 Undo performed a preliminary survey of their customers. The answers received were predominantly from the data management, networking, and electronic design automation industries.

Figure 8-2: The Undo Time Travel Debugging Study Results



New Bug Hunter Programs Are Emerging

In the meantime, we have observed the emergence of bug bounty hunter programs that are being used to identify bugs in the wild that need to be fixed, as seen in the following two examples focused on the OSS problem.

1. Google has launched a new program that will pay bounties for bugs found in its open source projects. You can earn up to $31,337 for finding a bug in Google's open source software. Google says that the Open Source Software Vulnerability Rewards Program (OSS VRP) covers various Chrome and Android code across the company's wider operations, which have resulted in over $38 million being paid out to more than 13,000 contributions, from a total of 84 countries. Furthermore, Google has pledged to invest $10 billion to improve cybersecurity among its own users and open source software consumers. Google says the OSS VRP focuses on "all up-to-date versions" of OSS stored in the Google-owned GitHub organization spaces, such as GoogleAPIs and GoogleCloudPlatform, though the "top awards" are reserved for the most sensitive projects, which Google sets out to be Bazel, Angular, Golang, Protocol buffers, and Fuchsia; a list that's expected to expand after the initial program rollout.
2. The SOS.dev initiative 'Secure Open Source Rewards' will help in preventing assaults on the software supply chain by incentivizing researchers to offer security upgrades to essential projects. This new initiative aims to reward developers and security experts that enhance crucial infrastructure using open source software. According to those who support it, the rewards initiative, which is 'Secure Open Source,' will cover more ground than bug bounty schemes at the current time. By encouraging academics and developers to make security changes, the program would "harden vital open source projects" and aid in protecting against application and software supply chain threats. Up to $10,000 is available for each bug found.

The Bleeding Edge For Automated Program Repair

Automated techniques for bug detection, mitigation, or prevention have a long history in computer science research. Programming languages and their type systems and compilers can warn programmers when they make certain kinds of mistakes or eliminate them entirely by design. Static analyses, sometimes built into integrated development environments or run at commit time, can flag problematic coding or architectural patterns or even, increasingly, find deep semantic errors. Dynamic self-healing techniques can enforce security or other correctness policies by enforcing control flow integrity, preventing code injection, or automatically sanitizing inputs. Such techniques can therefore catch and recover from errors at runtime, without either user or developer intervention.

By contrast, the techniques for automatic software repair generally aim to produce changes (patches) to the program source code to address the bug altogether (rather than find errors, help programmers avoid errors, or help systems dynamically recover from them).

Sometimes these goals can go hand in hand. For example, some static bug-finding tools increasingly provide the developers with pointers or suggestions to help them understand and fix the underlying problem; indeed, more quick-fix suggestions by bug-finding tools can lead to greater adoption. Similarly, compilers increasingly make suggestions to address flagged errors, and research techniques are being proposed to address more semantically complex bugs, as flagged by static techniques. Such approaches thus use a static bug-finding approach to find a flaw and then can use the static technique to automatically localize the bug and validate that a proposed patch addresses it (i.e., by determining that the static analyzer no longer flags the deficiency in question).

However, a larger preponderance of current techniques for automatic program repair are dynamic in nature. That is, these methods use failed tests or program crashes to demonstrate the existence of a glitch; the goal of the bug-repair process is to modify the program source code so that the test(s) now pass or the program no longer crashes. Other existing program tests are typically used to help the program-repair process avoid unwittingly breaking other desirable behavior, in the same way that continuous integration (CI) test suites help human programmers avoid doing the same in manually modifying their systems. Indeed, some proposed and currently deployed techniques are targeted at that use case exactly: repairing a program with respect to a failed CI test.

The successes of automated program repair, as the field stands today, have been significant. Successful techniques vary in terms of whether they address particular deficiency types or whether they aim to be more general to a wider variety of program properties that can be captured in a failing test. There has been tremendous progress in terms of enhancing generality of the techniques and scalability with respect to programs and search spaces. Modern research techniques of all stripes have reported successful results on programs of hundreds of thousands to millions of lines of code. Scalability to large search spaces (beyond simply to large programs) is important to allow the repair of complex, multipart bugs or programs that are significantly incorrect. Increasingly, such techniques are beginning to penetrate engineering best practice (e.g. Getafix, in Bloomberg).

9. ARTIFICIAL INTELLIGENCE (AI) AND MACHINE LEARNING (ML) IN SOFTWARE ENGINEERING

We can all remember the old saw about how the shoemaker's children had no shoes, because he was too busy satisfying his paying customers. Such was the situation in AI software development until recently when AI developers turned their attention to helping solve the complex problems of software development itself.

ML, Deep Learning, and Natural Language Processing (NLP) are frequently considered as three techniques within the larger domain of AI. When combined these techniques create some unprecedented possibilities to transform the software development process. With renewed interest in AI/ML and an emerging uniformity of software development processes (common repositories as well as CI/CD), industry is ripe for absorbing these ideas into the mainstream.

Back in the 1980's when the Microelectronics and Computer Technology Consortium (MCC) empirical studies of software engineers project was formed, we focused on the process of building new software systems from scratch. Today's predominant software development methods are more about combining/integrating existing components into even more complex systems. In this modern approach the difficulty has now shifted to understanding what the components do and how they interact/depend on each other. And so, working with pre-existing code bases that the development team probably did not write themselves is the new normal.

It is no surprise therefore, as R. Minelli, A. Mochi, and M. Lanza recently reported, that software developers now spend about 70% of their coding-related time in understanding the code, while the writing of code only accounts for about 5%.

Figure 9-1 Time spent understanding existing code

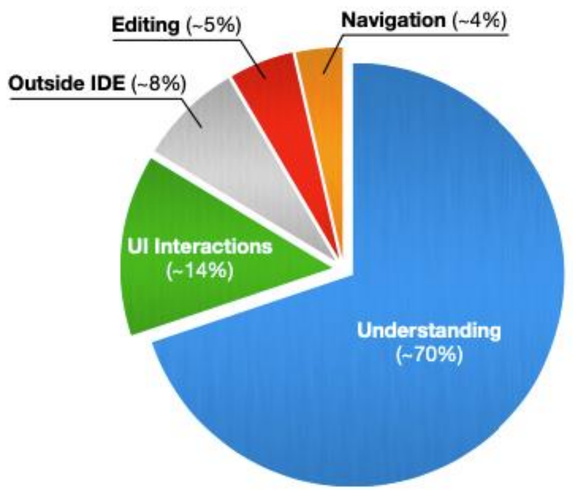


Fig. 3. How do developers spend their time?

Here are some areas of software engineering that we think AI/ML tools will have the biggest impact in the near future. However, these may only be realizable with a good solid ML focused development environment with the appropriate tools.

Precise Estimates

Software developers are notorious for seldom being able to provide good estimates on timelines and costs. AI trained on data from past projects can help provide more precise estimates so that teams can better predict the time, effort, and budget required. Now, if we can get the C-Suite to respect those estimates, the pressure to deliver too soon might be diminished, and failed projects minimized. This type of information can help an organization decide which projects to do and which not to. When you can accurately inform clients about software delivery timeframes, it increases customer satisfaction, retention and business reputation.

Establishing Quality Goals

When combined with better defined software quality standards (e.g. ISO 25000), new AI enhanced quality measurement tools will become possible. This will help organizations use past performances on software quality to learn from and then establish better project quality goals for new projects.

Error Management

When you provide past data and software analytics to an AI-powered programming assistant, it can learn from experience and identify common errors. If these are then flagged in the development process, it would reduce the need for rework. Machine learning can be used by operations teams in the post-deployment phase, as well, to proactively flag errors and uncover abnormalities by analyzing system logs.

Error management is responsible for most downtime in software development, especially if you offer Software-as-a-Service (SaaS) or Platform-as-a-Service (PaaS). With customers using your services round the clock, every minute of downtime costs money and negatively impacts an organization's reputation.

Understanding, Finding And Fixing Bugs

When an error (or deficiency) is detected in software, a developer has to understand it, find it, fix it and ensure it is fixed. This is a time-consuming process as shown in section 8 of this report. With artificial intelligence, you can semi-automatically detect and diagnose errors in the software without lots of human involvement. This process is more efficient and cost-effective, and leads to higher quality software.

Moreover, with the latest algorithms and advancements in AI and machine learning, developers and testers can predict and prevent errors automatically by searching through databases for known or learned patterns.

Finding Patterns In Code

AI can take static code analysis to the next level, using millions of lines of code to learn correct and incorrect programming patterns and then find those patterns in other code. This is especially important when integrating third-party code into a system.

This becomes much more effective when coupled with emerging knowledge bases of poor coding patterns as seen in the MITRE CWE, CVE repositories.

AI-Aided Automated Software Testing

Software testing is a crucial phase in software development, which helps ensure the quality of the product. Certain software testing must be repeated whenever source code is changed and repeating those same tests can be time-consuming and costly. Capturing and learning from that process is an AI strength.

There is a wide range of tools that employ AI for creating test cases and perform regression testing. These AI tools can automate testing and further ensure error-free retesting. Appvance, Functionize, and Testim.io are a few examples of AI and machine learning-based testing platforms.

Eggplant and Test Sigma are two popular AI-aided software testing tools that help software testers to write and execute automated tests to mitigate bugs and improve the efficiency of software code.

Software Security

Organizations across the world are using AI to capture security data and use Machine Learning to distinguish anomalous behavior from typical behavior. AI systems can be used to detect malware for cybersecurity, execute pattern recognition, and observe behaviors of malware before it enters the system.

AI/ML Development Assistants

A recent report found that AI-enhanced software development increased the productivity of a developer by 10 times, thus bringing novice level performance up to that of the expert.
Within the DevQualOps model, machine learning can shorten several processes, especially the testing of software. AI can run tests automatically, rather than having QA analysts run them manually. Not only does this save time but it ensures more scenarios are tested. AI is, in fact, critical to the quality assurance process as manual quality assurance has a high chance of error. AI enables a computer to do fast and accurate testing that reduces the failure rate and shortens the development process.

These new AI-based tools take things a step further, parsing and understanding all of those millions of lines of undocumented code out there and finding helpful snippets as you need them, without having to search for them.

A Better Understanding Of User Behavior

Machine learning algorithms can help to understand the user behavior and then deliver variable content by adjusting screen size, font size, buttons, and several other on-page elements.

Such personalized and dynamic responses can improve the user experience and it allows developers to make appropriate changes in the code by observing the real-time user interaction data.

AI and ML are implemented in Online Marketplace portals, where they can improve the software functionality, capture the user feedback, reduce the friction points, prevent abandoned carts, and increase the conversion rates.

Some Recently Emerging New Tools

The original OSS development environment Eclipse now has the capability that provides a core set of components for building applications that incorporate AI.

GitHub Copilot

GitHub hosts millions of projects, which, together, add up to billions of lines of code. GitHub, working with OpenAI's Codex machine learning model (a code-focused language model like the familiar GPT-3) has created a tool to build and train a service that works with your code editor to suggest next steps as you work. Calling it Copilot, GitHub describes it as an “AI pair programmer.” It is therefore a collaborative tool rather than a prescriptive one.

Copilot has been trained on the millions of lines of code in public repositories. Installed as a Visual Studio Code extension, Copilot works within the context of your current editor window, providing suggestions based on what you type and feeding back details on what you use. Your private code isn't used to train the service with new code samples. The only signals are the code you're using.

You shouldn't expect the code Copilot produces to be correct. For one thing, it's still early days for this type of application, with little training beyond the initial data set. As more and more people use Copilot, and it draws on how they use its suggestions for reinforcement machine learning, its suggestions should improve. However, you're still going to need to make decisions about the snippets you use and how you use them. You need to be careful with the code that Copilot generates for security reasons. It's impossible for GitHub to audit all of the code it's using to train Copilot. Even with tools like Dependabot and the CodeQL security scanner, there's a lot of poor-quality code out there exhibiting bad patterns and common bugs.

There are some interesting ideas in Copilot: how it takes your comments and turns them into code, or how it suggests the tests that can be used as part of a continuous integration/continuous deployment (CI/CD) process. Building AI into the dev and test parts of a CI/CD DevOps model makes a lot of sense, as it can help reduce the load on developers, letting them focus on code. But again, you still need to be sure that those tests are appropriate and that they provide the right level of code coverage. You're not limited to one solution at a time, as you can page through results in your editor, seeing what works best for you before you accept it.

Here some other interesting things that Copilot can do.

Microsoft BugLab

While there are dozens of tools available for static analysis of code in various languages to find security flaws, researchers have been exploring techniques that use machine learning to improve the ability to both detect flaws and fix them. That's because finding and fixing bugs in code can be hard and costly, even when using AI to find them.

Researchers at Microsoft Research Cambridge, UK have recently detailed their work on BugLab, a Python implementation of "an approach for self-supervised learning of bug detection and repair". It's 'self-supervised' in that the two models behind BugLab were trained without labelled data. This ambition for no-training was driven by the lack of annotated real-world bugs to train bug-finding deep-learning models. While there is vast amounts of source code available for such training, it's largely not annotated.

BugLab aims to find hard-to-detect bugs versus critical bugs that can be already found through traditional program analyses. Their approach promises to avoid the costly process of manually coding a model to find these bugs.

The group claims to have found 19 previously unknown bugs in open-source Python packages from PyPI as detailed in the paper, *Self-Supervised Bug Detection and Repair*, presented at the Neural Information Processing Systems (NeurIPS) 2021 conference.

Beyond reasoning over a piece of code's structure, they believe bugs can be found "by also understanding ambiguous natural language hints that software developers leave in code comments, variable names, and more."

Their approach in BugLab, which uses two competing models, builds on existing self-supervised learning efforts in the field that use deep learning, computer vision, and natural language processing (NLP). It resembles or is "inspired by" GANs or generative adversarial networks – the neural networks sometimes used to create deep fakes.

BugLab's two models include bug selector and a bug detector: "Given some existing code, presumed to be correct, a bug selector model decides if it should introduce a bug, where to introduce it, and its exact form (e.g., replace a specific "+" with a "-"). Given the selector choice, the code is edited to introduce the bug. Then, another model, the bug detector, tries to determine if a bug was introduced in the code, and if so, locate it, and fix it."

From the researchers test dataset of 2,374 real-life Python package bugs, they showed that 26% of bugs can be found and fixed automatically.

However, their technique flagged too many false-positives, or bugs that weren't actually bugs. For example, while it detected some known bugs, only 19 of the 1,000 reported warnings from BugLab were actually real-life bugs.

As for the 19 zero-day flaws they found, they reported 11 of them on GitHub, of which six have been merged and five are pending approval. Some of the 19 flaws were too minor to bother reporting.

Facebook's Getafix-

For the last few years Facebook has been using an internally developed tool called Getafix, which they claim contributes to the stability of apps that billions of people use.

They claim that of all the warnings fixed by Facebook engineers since the Getafix service was rolled out, 42% were fixed by accepting the fix suggestion, and, in 9% of the cases, engineers wrote a semantically identical fix. They have successfully started automating the discovery and application of "lint" rules. Changes made in response to code review are often fixes to common antipatterns that were pointed out by a reviewer, and finding and fixing these antipatterns can be baked into a lint rule.

In their view the most promising but relatively untapped opportunities for using ML pertinent to aspects of the team and production states are.

- *Codereview*: while widely regarded as essential for maintaining software quality, manually reviewing code is a significant time commitment for software engineers. ML techniques can help automate routine code reviews (such as formatting and best coding practices). More ambitiously, perhaps ML can automatically resolve a routine code-review comment.
- *Assessing the risk of a code change*: In principle, any code change increases the riskiness of an application. Arguably, the entire testing and verification pipeline exists essentially to reduce this risk. Can ML-based techniques be designed that provide a quantitative assessment of the risk of a code change, complementing

the usual testing and verification pipeline? Advances here will impact both testing (by prioritizing tests related to riskier changes) and release management (by carrying out additional quality control for riskier code releases). By comparison, techniques for assessing the impact of a change take a binary view of affectedness and, due to the limitations of static analysis, often would be overly pessimistic in their assessment.

- *Troubleshooting*: For widely deployed applications, customers send their feedback implicitly (telemetry or crashes) and, sometimes, explicitly by sending comments. The volume of this feedback can be huge. This is another area where ML can help in multiple ways: not only in triaging these reports, but clustering them to identify common issues, finding important clues from telemetry logs and code changes that could be connected to the issue at hand.

Kite

Kite, by suggesting context-aware reusable code, can help a developer decrease keystrokes by 47%. It was trained on models that have gone through more than 25 million files and, as a result, can offer multi-line suggestions. Kite is compatible with 12+ languages that include Java, PHP, HTML/CSS, Javascript, Typescript, Kotlin, Ruby and Python. **Codota is similar.**

Visual Studio IntelliCode

IntelliCode is from Microsoft and comes integrated with Microsoft's IDE named Visual Studio. In Visual Studio, it supports C# and XAML, while it is compatible with Java, Python, JavaScript, and TypeScript in Visual Studio Code. This AI code completion tool received its training from the codes of half a million of GitHub's open-source projects. Therefore, it can guide you with smarter suggestions considering the current code and context. To do so, it takes assistance from variable names and positions, the IntelliSense list, Libraries that you use, and functions in nearby code. While this tool will show you suggestions in alphabetical order by default, you can always toggle between the options. Its whole line code completion feature, available in the 2022 version of Visual Studio, indicates the next chunk of code based on your gray text inline prediction.

Next Steps

What's missing are the AI-based deficiency prevention tools that train on known patterns (e.g. CVEs, CWEs, etc.) to catch them in real time as software developers are writing the code that creates those deficiencies/weaknesses, thereby preventing them from ever getting into the development stream of new code. When these tools finally arrive the CPSQ will plummet, and we can turn our attention to remediating the growing TD.

10. CONCLUSIONS, RECOMMENDATIONS, AND NEXT STEPS

Conclusions

The key US economic conditions that frame the context for this biennial report are:

- A projected GDP for 2022 of $23.35 trillion, a roughly 2% rise since 2020
- A cumulative inflation rate of 15% over the 2 year period
- A small 4% growth in the IT labor base over those 2 years, and
- The number unfilled IT jobs sitting at about 300,000 as of the end of August.

In this 2022 update report we estimate that the cost of poor software quality in the US has grown to at least $2.41 trillion[1], but not in similar proportions as seen in 2020. The accumulated software TD has grown to ~$1.52 trillion[1]. These are primarily due to:

- The huge rise in cybercrime costs to $1.44 trillion in 2022, which accounts for most of the rise in CPSQ, and
- The shortage of qualified software engineers along with the lagging use of available tools accounts for the rise in TD, largely because deficiencies are not getting fixed at the same rate as in 2020.

Although the CPSQ and TD have risen significantly over the series of our three reports (the problem), so have the developments in the technology/practices to remediate those problems (solutions).

IT IS POSSIBLE THAT THE TREND IN OVERALL CPSQ WILL FLATTEN OVER THE NEXT DECADE IF ORGANIZATIONS WILL ADOPT THE RECOMMENDATIONS THAT WE HAVE PUT FORWARD IN THIS SERIES OF REPORTS. We hope that the solutions suggested herein become more widely adopted into the mainstream of software conception, development, production and evolution.

Recommendations

In addition to the broad recommendations of our previous reports, we add the following more specific recommendations for software development and IT organizations:

- Use the software quality standards, related measurements and tools that are emerging
- Analyze and assess the quality of all 3rd party/OSS components to be included in any system. Monitor them closely in operation. Apply patches in a timely fashion.
- Avoid DevOps and CI/CD models that do not include continuous quality engineering best practices and tools. Adopt DevQualOps instead.
- Integrate continuous TD remediation into your SDLC
- Invest in the professionalism, knowledge and tooling of your software engineers, and
- Consider having your developers certified for knowledge of the critical code and architectural weaknesses in ISO/IEC 5055 (when OMG makes its "Dependable Developer' certification test available in late 2023 or 2024).

Dealing With The IT Job Market Shortage

According to the US Bureau of Labor Statistics (BLS) the IT positions listed below are selected examples that are projected to grow and pay well above average. Due to the current shortage these crucial positions will be more difficult to fill – especially in the first three categories below.

Software Developers, Quality Assurance Analysts, and Testers
Software developers design computer applications or programs. Software quality assurance analysts and testers identify problems with applications or programs and report defects.
2020 median pay: $110,140 per year
Typical entry-level education: Bachelor's degree
Number of jobs, 2020: 1,847,900
Projected growth, 2020–2030: 22% (Much faster than average)
Occupational openings projected, 2020–2030 annual average: **189,200**

Computer Systems Analysts
Computer systems analysts study an organization's current computer systems and find solutions that are more efficient and effective.
2020 median pay: $93,730 per year
Typical entry-level education: Bachelor's degree
Number of jobs, 2020: 607,800
Projected growth, 2020–2030: 7% (About as fast as average)
Occupational openings projected, 2020–2030 annual average: **47,500**

Information Security Analysts
Information security analysts plan and carry out security measures to protect an organization's computer networks and systems.
2020 median pay: $103,590 per year
Typical entry-level education: Bachelor's degree
Number of jobs, 2020: 141,200
Projected growth, 2020–2030: 33% (Much faster than average)
Occupational openings projected, 2020–2030 annual average: **16,300**

Computer and Information Research Scientists
Computer and information research scientists design innovative uses for new and existing computing technology.
2020 median pay: $126,830 per year
Typical entry-level education: Master's degree
Number of jobs, 2020: 33,000
Projected growth, 2020–2030: 22% (Much faster than average)
Occupational openings projected, 2020–2030 annual average: 3,200

Web Developers and Digital Designers
Web developers create and maintain websites. Digital designers develop, create and test website or interface layout, functions and navigation for usability.
2020 median pay: $77,200 per year

Typical entry-level education: Bachelor's degree
Number of jobs, 2020: 199,400
Projected growth, 2020–2030: 13% (Faster than average)
Occupational openings projected, 2020–2030 annual average: 17,900

Please see the BLS Occupational Outlook Handbook for others and more detail.

Higher demand for those professionals will continue over the next decade due to the increase in telework and hybrid work arrangements, expanded tele services and enhanced cybersecurity measures to protect information.

Overall, the job market for tech talent in 2022 remains strong. In August, the unemployment rate for tech occupations in the US stood at 2.3%, according to the Computing Technology Industry Association (CompTIA), significantly lower than the US unemployment rate of 3.7% that month, which is itself low by historical standards. There are an estimated 8.7 million tech workers in the US, according to numbers CompTIA released earlier this year.

In total, more than 118,000 people have lost their jobs in tech this year, according to Layoffs.fyi, a site that tracks publicly reported job cuts in the industry. But all of those laid off have excellent prospects in either established companies or in startups.

It is also not yet clear whether the massive mid-November, 2022 layoffs in the (un)social media technology industry will have any impact on this shortage of software professionals.

Next Steps

Our next report is tentatively planned for 2024, when hopefully some of the solutions identified in this report will catch up with the problems, and show up in a positive change to the CPSQ trend. Our next report will probably focus on the trustworthiness of critical infrastructure systems, in healthcare, elections and energy distribution.

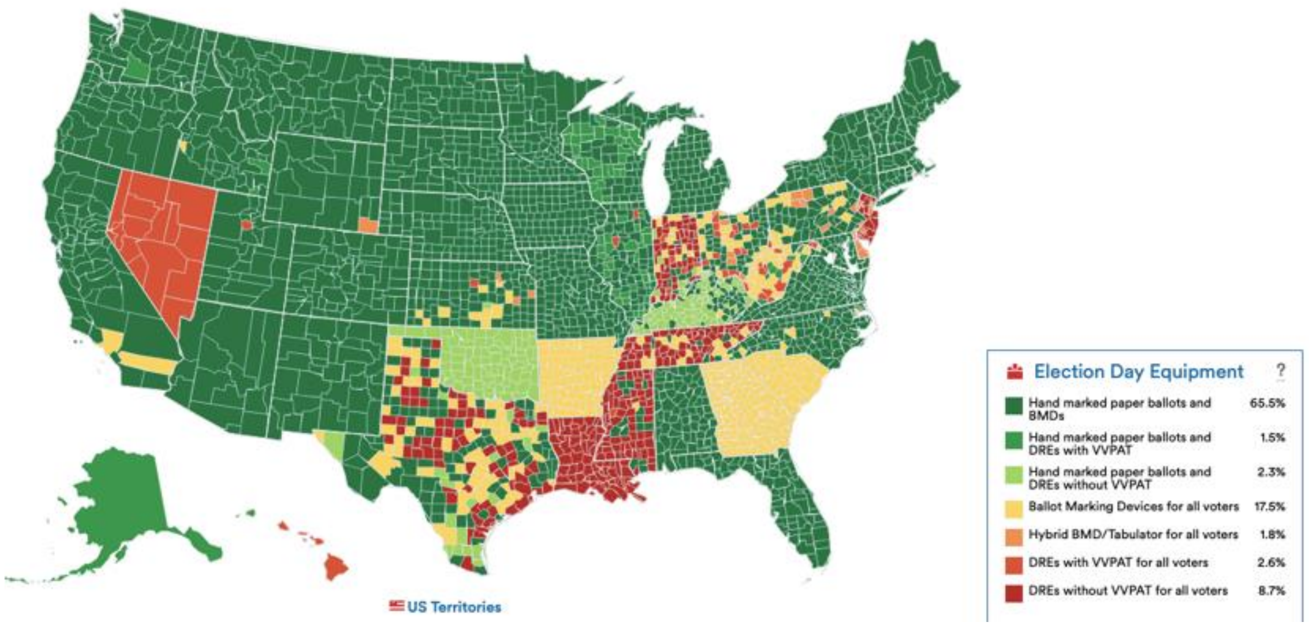
We expect this particular area to be of growing interest in 2024.

A Focus On The Critical Infrastructure Of Election Systems Technology

In the area of voting systems technology, we have come a long way since the days of “hanging chads”. For two decades, the rise of voting technology has helped to automate a process that was in dire need – thus allowing millions of voters to vote without adding to the human effort of administering that process. Unfortunately, as a side effect these systems have become a focus of intense political debate.

Because our election system is actually a collection of independent voting systems the range and diversity of election systems go from “totally paper” to semi-automated. For example, in Texas there are 254 different county level systems in use. In much of the country, when somebody votes, they just fill out a paper ballot, which is typically fed through a tallying device called an optical scanner. Elsewhere, some voters use fully digital setups, called direct recording electronic systems, that sometimes use the computer to both mark and tally the votes. Elsewhere, hybrid and home grown systems are used. We know that at least 30% of the votes cast in the 2020 general election were on some kind of machine, as opposed a hand-marked ballot. This diversity of election technology is seen in the figure below.

Figure 10-1 The Diversity of Election Technology in the US: 2020 General Election


US Territories
Election Day Equipment ?
Hand marked paper ballots and BMDs 65.5%
Hand marked paper ballots and DREs with VVPAT 1.5%
Hand marked paper ballots and DREs without VVPAT 2.3%
Ballot Marking Devices for all voters 17.5%
Hybrid BMD/Tabulator for all voters 1.8%
DREs with VVPAT for all voters 2.6%
DREs without VVPAT for all voters 8.7%

Today, the voting machine market is dominated by three major vendors: Election Systems & Software, Dominion Voting Systems, and Hart InterCivic. According to one estimate, the entire industry generates approximately $300 million in revenue annually.

Most of their products contain a semi-automated ballot-marking device (BMD). While specific designs vary, BMDs have a computer touchscreen for voters to make their selections. The machine then prints out a paper ballot that can be fed into a scanner. Unlike hand marked paper ballots, BMDs have the ability to accommodate every voter using a variety of accessibility devices — including the ones who can't see, handle paper, or even touch a screen. The machines have proliferated since 2002, when Congress passed the Help America Vote Act.

Voting machines have other advantages over paper ballots: They can offer multiple language options, support larger jurisdictions that need thousands of different ballot types, and ensure that voters don't inadvertently miss a race or make a mistake that disqualifies their ballot. It's not clear how big a problem that might be.

The Software Security Issue

Advocates say the electronic voting systems can be relatively secure, improve accessibility, and simplify voting and vote tallying. But, critics have argued that they are insecure and should be used as infrequently as possible. There are hackers hard at work on both sides.

Semi-automated voting brings fears that someone could tamper with the machines and manipulate the results. And, some experts say, the vendor companies' behavior has done little to inspire public trust. These critics say that the software in BMDs is complex, often poorly organized, and extremely long, making it easier to insert code that goes undetected. But there is no empirical data to back up or deny that assertion.

Because the races and candidates change every election, a new ballot design must be uploaded before every contest, offering another opportunity for malicious code to slip in. And because voting is done anonymously, it's impossible to link a specific ballot to the person who cast it after the fact.

In response to such concerns, voting machine companies have acknowledged that their equipment may have vulnerabilities. But, they say, nearly all the machines leave a paper trail that can be audited, making it possible to catch incidents during certification.

Amid these concerns, a handful of innovators are trying to create a voting machine that's easy to use, based in open-source software (OSS), and significantly more difficult to hack than existing models. But what we have recently seen in the vulnerabilities in OSS that is an assertion to be tested.

One example of a research prototype of such a machine is seen in one of the recent papers published in the Communications of the ACM in November, 2021. As with most research prototypes, this model has yet to be subjected to rigorous external testing by unfriendly users.

Hopefully by 2024 we will have some experimental data on the software quality in these types of prototype voting machines.

Footnote 1 – See Appendix B for the detailed cost estimation methodology used.

11. ACKNOWLEDGEMENTS

About the author. Herb Krasner is a retired Professor of Software Engineering at the University of Texas at Austin. He has been an industry consultant for five decades; helping organizations baseline and improve their software development capabilities. He is an active member of the CISQ Advisory Board and is well-known for his previous research in the empirical studies of software professionals, the ROI of software process improvement, and the cost of software quality. He can be reached at hkrasner@utexas.edu.

Grateful thanks to our technical review team for their most helpful review comments and contributions to the drafts of this report: Anita D'Amico, Marco Puebla, Greg Law, and Laila Lotfi. Special thanks to my long-time friend, collaborator and chief reviewer, Don Shafer, Athens Group founder and Technical Fellow – with whom I have worked on many important initiatives since the mid-1980's. Much thanks to our project administrator, Katie Hart; and to my buddy Bill "Tex" Curtis, Executive Director, CISQ, for his five decades of friendship, professional collaboration, and support. Thanks to my wife Judy for her professional editing assistance, plus 51 years of love, support, and putting up with me.

About CISQ. The Consortium for Information & Software Quality™ (CISQ™) develops international standards to automate software quality measurement and to promote the development and sustainment of secure, reliable, and trustworthy software. Through their work, industry-supported standards have been developed to measure software size, structural quality, and TD from source code. These standards are used by many IT organizations, IT service providers, and software vendors in contracting, developing, testing, accepting, and deploying software applications.

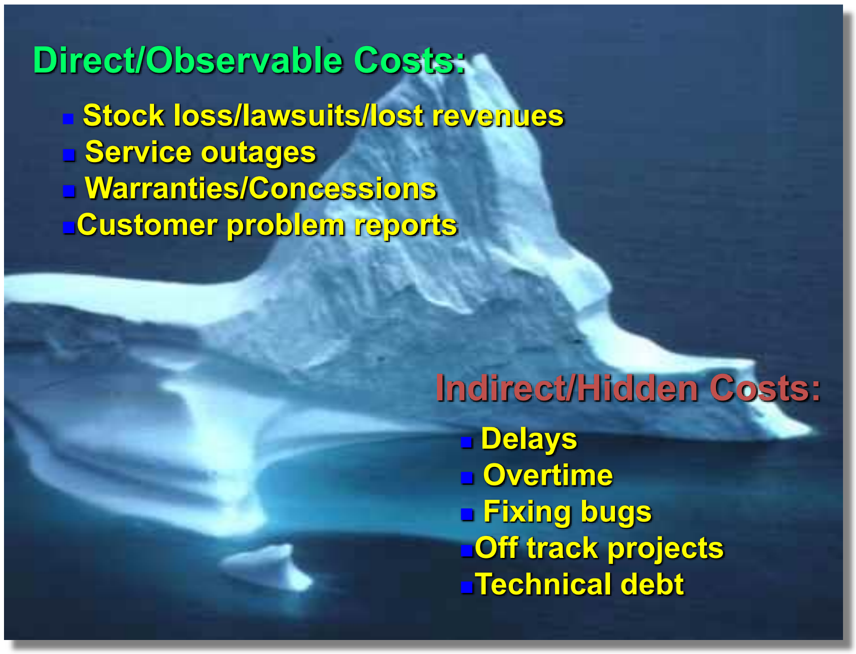
2022 Report Sponsors

Synopsys
Synopsys is the #1 electronic design automation company that focuses on silicon design and verification, silicon intellectual property and software security and quality. Their technology is present in self-driving cars, artificial intelligence, and internet of things consumer products. They have invested over $1 billion into building the ultimate software security solution. The Synopsys Software Integrity Group focuses on software security and quality. They provide static analysis, software composition analysis, and dynamic analysis solutions that enable teams to quickly find and fix weaknesses, vulnerabilities and defects in proprietary code, open source components, and application behavior. Ranked as the leader in Gartner's Magic Quadrant for Application Security Testing, with a combination of industry-leading tools, services, and expertise, Synopsys SIG helps organizations optimize security and quality in DevSecOps and throughout the software development life cycle.

Undo
Undo is the time travel debugging company for Linux. They equip developers with the technology to understand complex code and fix bugs faster. Developers spend far too much time figuring out what code actually does – either to understand other people's code or to find and fix bugs. Debugging can be especially time-consuming when software failures cannot be reproduced. Time travel debugging solves this problem by making bugs 100% reproducible. By bringing time travel debugging to CI and System Test, Undo's LiveRecorder enables developers to save time diagnosing the root causes of new regressions, legacy bugs, and flaky tests. Thousands of developers across leading technology firms including SAP, Juniper Networks, and Siemens use LiveRecorder to improve developer productivity, development velocity, and software quality.

APPENDIX A: CPSQ 2020 REPORT SUMMARY

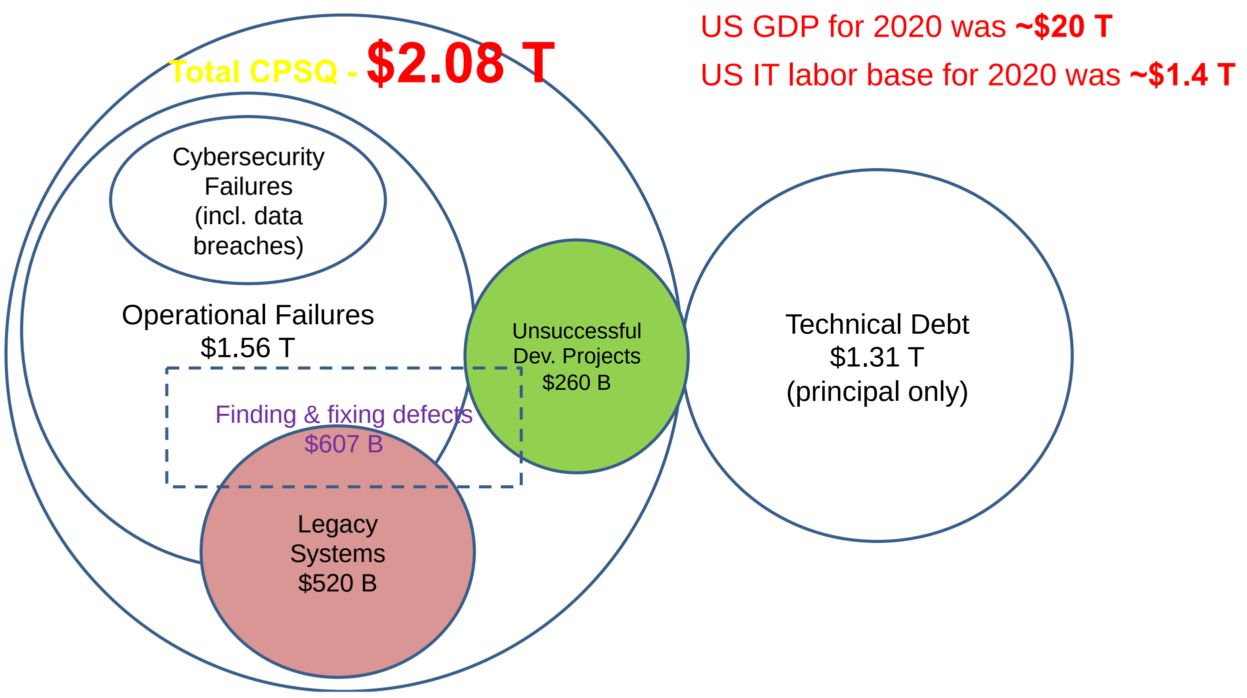
The primary purpose of our series of reports has been to inform and inspire our readers to seek CPSQ knowledge within their own organizations, by first exposing the size of this mostly hidden problem is. In our 2018 report we stated that the hidden costs of poor software quality were 6 to 50 times the observable costs. This may have helped to make are readers more aware of what could be made observable.



The 2020 report built on some of the basic 2018 concepts of what is software quality, what is the cost of software quality model, and the discussion of good vs poor software quality.

In our 2020 report, we elaborated many of the publicly known failure reports to emphasize the sheer magnitude of the poor software quality problem. We laid out most of the strategies, tactics, models and best processes/practices that might be used to tackle the problem via a coherent approach that organizations could use.

In the 2020 report we showed a summary of the estimates of the cost of poor software quality in the US for that year as seen in the figure below.


Total CPSQ - $2.08 T
US GDP for 2020 was ~$20 T
US IT labor base for 2020 was ~$1.4 T
Cybersecurity Failures (incl. data breaches)
Operational Failures $1.56 T
Finding & fixing defects $607 B
Unsuccessful Dev. Projects $260 B
Legacy Systems $520 B
Technical Debt $1.31 T (principal only)

We were able to construct this result using a unique analysis, synthesis and extrapolation of 88 existing sources of available online information, mixed with expert knowledge about software and its quality. A summary of each major category from the 2020 report is presented below.

Operational Failures: $1.56 Trillion

As well as citing specific examples, like Knight Capital, and SolarWinds, we identified that the Tricentis Software Fail Watch, 5th Ed. reported 606 major software failures from 2017, causing a total loss of $1.7 trillion in assets at 314 companies. This averaged out to $2.8 billion per failure. This led us to observe that cybercrime was one underlying driver that was escalating rapidly.

We identified the trends that magnify the impact of software flaws, driving failure costs up:

- 100+ billion new LOC produced worldwide each year -> 25 bugs per 1000 LOC injected on average
- 96 zettabytes of digital data now stored (up from 16 in 2016)
- Growth of cybercrime – ransomware in US cost $9B; $20B worldwide in 2021
- Increasing Digital Transformation: spreading the effects of a software malfunction across the entire value chain.
- Growth of Systems of Systems: expanding complexity exponentially and concealing the triggers for huge failures in a thicket of cross-system interactions.
- Increased Competition: especially online, has prioritized speed-to-business over operational risk and corrective maintenance costs
- a huge gamble for systems not designed to expect and manage failures.

Our broad recommendations to deal with potential operational failures moving forward was to:

- Prevent bugs, flaws, weaknesses, vulnerabilities from being created and fielded
- Find and fix bugs early
- Measure quality
- Adopt high quality development practices
- Analyze potentially flawed components (e.g. OSS)

Unsuccessful Projects: $260 Billion

By examining the IT Project Outcomes from the series of CHAOS reports and extrapolating the trends across them, we observed that the industry held steady in this area with

- Approximately 20% of projects outright failing
- Approximately 35% of projects succeeding
- And the remaining 45% in the challenged category

Our broad recommendations were to:

- Avoid huge projects: For projects of large size (10^4 FPs) and above, low-quality projects are 5-6X more likely to be cancelled then high-quality projects.
- define what quality means for a specific project and then focus on achieving measurable results against stated quality objectives
- use known best practices & tools for achieving high quality
- don't compromise quality for speed to operation

Legacy System Problems: $520 Billion

After decades of operation, these systems may have become less efficient, less secure, more brittle, incompatible with newer technologies and systems, and more difficult to support due to loss of knowledge and/or increased complexity or loss of vendor support. We noted that these systems typically:

- consume 70-75% of the total IT budget
- account for 80% of the total cost of ownership

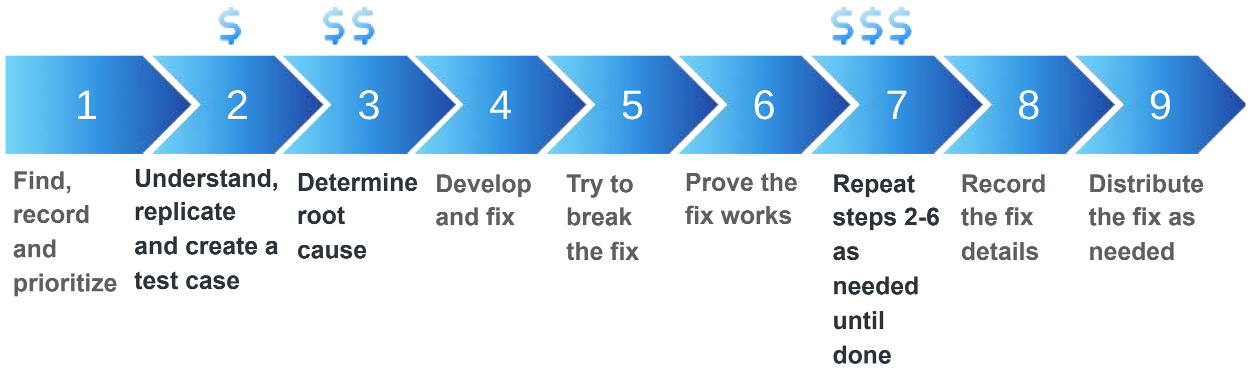
The modernization approach to be used depends on the priority of problems to be solved – functionality, performance, obsolete technology, inflexible architecture, loads of TD.
Several strategies were identified (e.g. containerization)

Our broad recommendations were that:

- All these strategies are enabled by overcoming the lack of understanding and knowledge of how the system works internally.
- Any tool which helps identify weaknesses, vulnerabilities, failure symptoms, defects and improvement targets is useful
- Benchmarking the health status of a legacy system is a good starting point.
- Detailed blueprints of system connectivity are useful for modernizing architectures that have degraded over time.

Finding and Fixing Bugs: $607 B

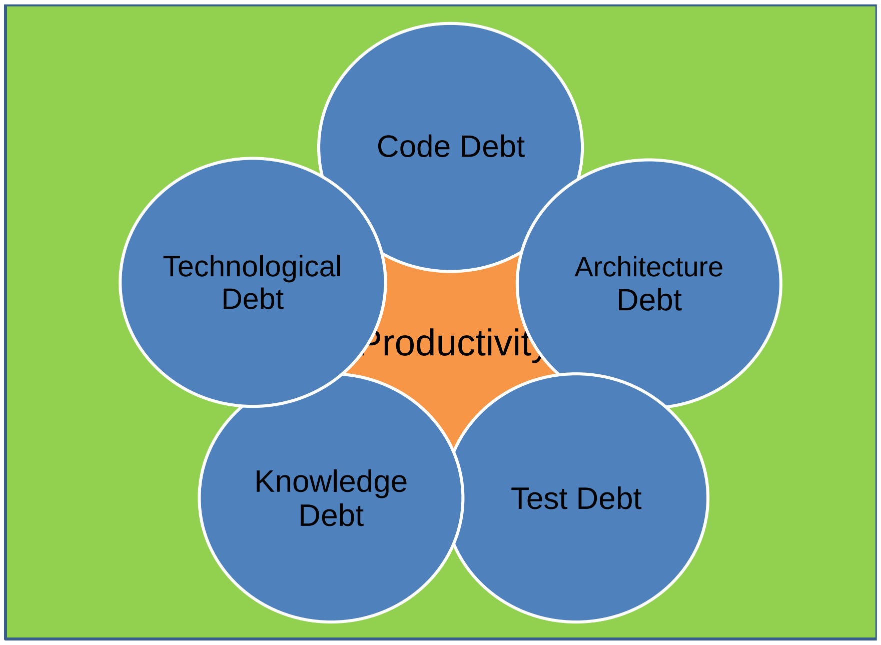
This most important area for software quality engineering underpins all of the other areas that we have described above. Based on our previous empirical studies we were able to describe where the relative effort is spent in this process, as seen in the following diagram.


$
$$
$$$
1
2
3
4
5
6
7
8
9
Find, record and prioritize
Understand, replicate and create a test case
Determine root cause
Develop and fix
Try to break the fix
Prove the fix works
Repeat steps 2-6 as needed until done
Record the fix details
Distribute the fix as needed

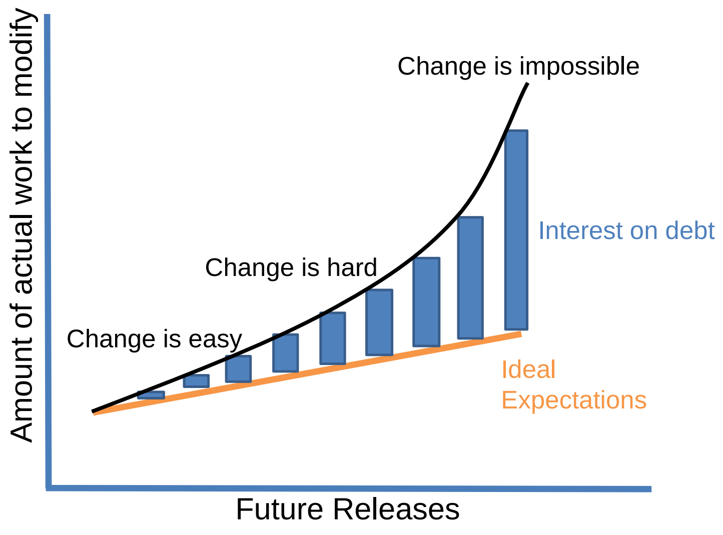
Our broad recommendation was to prevent problems first and then focus on where the $$$ are spent in the above process model to reduce CPSQ and improve overall quality and productivity. Given the importance of this process, we elaborate on this subject in this 2022 report.

TD: $1.31 Trillion + Interest

We described in 2020 that there are actually many types of software TD.



And how the impact of TD grows over time if left untreated.



We noted that this trend is exacerbated by the sheer size of the code base and how fast that is growing. Specifically, that

- ~1.655 trillion LOC exist worldwide and 513 billion in the US.
- code growth is now ~100 billion new LOC per year, or ~7% growth per year.

The key issue is now how do you identify and then manage software TD?

Overall Recommendations for improvement in practice

We reported our overall recommendations for those organizations that wanted to improve their cost of poor software quality profile, which we suggested should be done as a holistic organizational approach involving the following levels of an organization.

Leaders/C-Suite level

- Establish quality as a 1st-class citizen -> security+
- Ask better questions: externally and internally
- Measure software quality & CPSQ in your organization

Teams/projects

- Strive for high performance
- Use best practices & tools
- Define & track quality objectives
- Avoid arbitrary and unrealistic schedules or constraints

Individuals

- Learn and grow a disciplined approach
- Don't be afraid of quality metrics
- Use existing knowledge sources of bug pattern and structural quality flaws

APPENDIX B: CPSQ ESTIMATION METHOD

As we have stated before and continue to assert here, most organizations do not yet collect and report their cost of poor software quality numbers. For example, in one 2017 study of IT executives, 35% of those who were surveyed said they had no idea how much IT system failures were costing their business.

Operational failure costs are by far the largest category of CPSQ.

It is helpful to think of IT operational failure costs in terms of direct and indirect costs.

Direct Costs

Direct losses will fall into two general categories:

- Loss of an application or service. This can be more or less severe, depending on what has failed. For example, a series of unplanned outages can kill a business.
- Loss of data. Losing data can have an even bigger impact on a business because data loss can be permanent. This can also have financial and even legal implications beyond the direct losses. Ransomware has been particularly damaging in that regard.

Some data exists to help estimate direct costs.

- According to a 2008 study by IBM Global Services, the average revenue cost of an IT systems outage was $2.8 million per hour. Due to inflation that number today would be $3.88 million per hour.
- According to research by KPMG, 48% of companies say that more than 24 hours of downtime is unacceptable. And for an additional 24%, even a 2-hour outage will damage their business.
- A Dunn & Bradstreet survey showed that 59% of Fortune 500 companies experience 1.6 hours of system downtime per week or more.

Indirect Costs

Beyond the direct costs of an IT system failure, there are additional costs – financial and otherwise – that can significantly impact a business. In some cases, these can be much higher than the direct costs. But these are much harder to estimate.

Impact to other projects

- According to the Harvard Business Review, 27% of IT projects run over budget, and 70% are not completed on time. At first glance, this might seem like any other sunk cost – the cost of doing business. But it's not that simple. When an IT failure causes a delay in one or more projects, the issue has a compounding effect on a business. Money that's spent to complete the project is money that can't be spent on something else. Employees have to be pulled off other projects to meet the unplanned labor needs. These are all extra costs that won't be seen immediately, but which add up over time.

Reputational Damage

- Lost sales aren't just a one-time cost. To truly understand the impact of a lost sale, you also have to consider the lifetime value of any lost customers. In some cases, this can be several times the immediate, direct cost.
- Reputational damage is a major factor in the healthcare and financial services industries or any industry where access to a customers' personal information is at risk. If this information is leaked due to a security breach, the loss of public trust can cause the loss of a lot of business down the road.

Regulatory and Compliance Impact

- If an IT failure causes a company to miss a customer deadline or fail to follow through on contractual obligations, those costs will have to reimbursed to those customers. Worse, failure to meet the conditions of an SLA can land a company in trouble with regulators, resulting in significant fines.

Remediation Costs

- Beyond the direct cost of finding and fixing the problem, following an IT system failure extra work is often necessary to make up for that failure.
- These costs can become even higher if the organization's reputation has been damaged, in which case an entire marketing campaign might be needed just to repair the damage.

Morale Impact

- Beyond the loss of revenue and reputation, there's also the question of how a system outage affects the employees of an organization. If the system is one that is used internally, it has a direct effect on everyone involved. For an IT manager in particular, even a brief outage can feel like a personal failure. Executives can feel under pressure to find someone to blame, which can further erode morale.
- A system outage also wears on anyone who has to help fix the damage. People who need to work overtime are missing time with their families, or giving up time on their hobbies. If this happens too often, the workforce will lose faith in their leadership, and top employees will start applying for jobs with competitors.

Bottom Line on Operational System Failure Costs - *it depends*.

The most significant factors in determining costs is the type of industry and the severity of the failure. For the media sector, the average hourly loss is $90,000 per hour. But for large financial brokerages, the losses amount to a staggering $6.48 million for every hour of downtime. Energy, financial services, manufacturing, and telecommunications appear to be the industries with the highest IT downtime costs. Risks are particularly high for companies whose business requires a high rate of customer responsiveness.

Basis of Estimate

Assuming that our cost estimates in 2020 were close, we can project those forward based on the changes in economic conditions. Due to just inflation at 15% cumulative, our $1.56 trillion estimated in 2020, jumps to $1.8 trillion in 2022. However, we believe that the actual cost is much higher than our conservative estimate.

In the growing area of cybercrime alone, if we assume that the costs of cybercrime are proportion to those of the world's economy then the US would have a 24% share based on relative GDP size. The cost of cybercrime in the world in 2022 was estimated at $6-7 trillion. Therefore, the US share would be ~$1.44 trillion. That would make cybercrime ~80% of the total cost of all operational failures. It is doubtful that all the other types of operational failures amount to just 20% of that total.

In all the other areas of CPSQ we simply assumed that there was no growth, due to the shifting of scarce resources into dealing with failures and deficiencies (in spite of a cumulative inflation rate of 15%).