In partnership with

CyberRisk ALLIANCE

ICIT

The Institute for Critical Infrastructure Technology (ICIT)

How to Build a Compelling Business Case for a Software Security Program

By Jim Routh, Board Member

Executive Summary

Creating a compelling business case to invest in software resilience is more about economics than risk probability. The primary message for stakeholders is that eliminating and reducing defects in the software as it is assembled and modified increases productivity for software developers.

Enterprises spend heavily on software development and maintenance each year, and lowering that cost is highly attractive to the C-suite. Investing in better tools to eliminate and fix defects earlier results in increased cost avoidance and productivity. The up-front investment is made over three years, while the annual benefits carry forward each year. Any CFO, CEO or CIO will be receptive to lowering the cost of software ownership, so the company has the capital to invest in new capabilities that potentially drive revenue.

Economic rationales will always win over risk probability projects with business stakeholders because there is no way to apply attribution for software defects. Website and mobile software defects are rarely attributed to the enterprise, and the criminal exploitation of the vulnerabilities often lags the cyber intrusion by many months.

The right approach is to hold stakeholders accountable for producing and maintaining resilient software by measuring the defect density of the software's code during development. Defect density is the quantity of code assembled divided by the number of risky defects identified.

The business case for a software security program is that productivity can be improved by fixing security defects during the process of creating and maintaining software. This will have a significant impact on the business by producing more resilient software with a lower cost of ownership and fewer customer cyber incidents.

ICIT | CyberRisk ALLIANCE

Contents

Understanding Software Resilience

Enterprises of every size often struggle to justify the necessary investment in tools and capabilities to improve software resilience. This challenge is not new, nor is it a direct result of the evolution of the software supply chain attack surface, which has undergone significant changes in the past several years. For cybersecurity professionals seeking substantial improvements in software resilience, which is defined as quality software with limited exploitable security vulnerabilities, this paper provides a blueprint for creating a compelling business case to fund the investments needed to acquire tools (IAST, DAST, CA, pipeline instrumentation, etc.) and encourage developers to adopt resilient practices.

Throughout my 20-year career as a cybersecurity leader, I designed and implemented seven software security programs for large enterprises using "software security" instead of "application security." This term is inspired by Dr. Gary McGraw's book, "Software Security: Building Security In," first published in 2006 [1]. Software security encompasses the development of internal applications and the acquisition and assembly of all software components in the enterprise. I suspect that Gary and I may be among the few continuing to use this term today, given that application security is more common with cybersecurity professionals.

Of the seven software security programs I implemented as a CISO, I'm excluding the first one at American Express. At the time, I faced a steep learning curve regarding leading practices in security and how to achieve software resilience. I focused on a mandatory, online training class for all developers that was implemented without additional controls. I consider this initial effort a mixed bag, where I probably learned more than the developers. In each of the other six companies, I took a more thoughtful approach to building the business case for software resilience, resulting in an effective and mature software security program that yielded significant business benefits.

These programs were in large enterprises that required significant monetary investment in tools, methods, and techniques designed to change how developers work. Notably, I never encountered difficulties in justifying these expenses to my key stakeholders, and I received full funding for each program. If you aim to develop a compelling business case for funding a software security program, consider a similar approach to what I used.

Reduced Risk Is a Byproduct, Not the Driver

When creating your business case for software security capabilities and controls, avoid the common mistake of basing it solely on reducing cyber risk to the enterprise. Attributing web or mobile application security incidents to the enterprise is challenging, if not impossible. For instance, when a cybercriminal uses cross-site scripting (XSS) to inject malicious scripts into a victim's browser and steal data, the digital consumer is unaware of the root cause: a vulnerable web application. Even if the cybercriminal is apprehended months later, the risk of attribution to the enterprise is minimal.

The CFO of a large enterprise may recognize that spending money to reduce the probability of a cyber incident has a limited return on investment due to the inability to provide attribution to your vulnerable website. This reality may seem a bit callous, but it's accurate. While reducing the impact on digital consumers remains a positive outcome, you should consider it a byproduct of a software security program rather than the primary business driver. The rationale for investing in software quality is decreasing the unit cost of developing and supporting applications year over year. This approach will get the attention of every CFO simply because a lot of money is spent maintaining software every year.

Benjamin Franklin's life was a testament to his wit and wisdom, as illustrated by his oft-quoted phrase, "Time is money." This phrase highlights the value of time as a precious resource. In enterprises that rely heavily on software, this adage holds particularly true. Creating and assembling software components requires significant time from developers, driving up software development and support costs. Developer time accounts for a substantial portion of an enterprise's balance sheet, whether the developer is an employee, contractor, or works for a software provider.

As a cybersecurity leader, it's essential to leverage this concept as the foundation of your business case. Developer time is a tangible cost to the enterprise, and it's crucial to recognize this when advocating for advanced software security capabilities.

The Pillars of a Software Security Business Case

TABLE 1: Key Components of a Software Security Business Case

	Pillar	Cost savings	Productivity gain	Cost avoidance	Comments
1	Lowering the cost of defect fixes by identifying them and fixing them in the development cycle and not after production	✓	✓	✓	Cost savings occur when contractors finish earlier. Productivity gain comes from fixing defects in development. Cost avoidance occurs when it takes less time to fix defects before production
2	Fixing defects post-production reduces help desk call volume for resetting compromised passwords and reduces brand erosion from insecure websites			✓	Help desk call volume and password reset infrastructure usage is lower when customers credentials are not compromised due to software defects in either web or mobile applications.
3	Higher quality software costs less to support			✓	A % of annual maintenance cost can be avoided when the code defect density is lowered

By emphasizing these three components, you can create a compelling business case for investing in advanced software security capabilities that yield significant cost savings, productivity gains, and improved customer satisfaction. As you can see, decreasing the probability of a cybersecurity incident is the byproduct, but not the driver.

The business case for investing in these capabilities has three key components, as listed in TABLE 1:

1. **Reducing the Cost of Defect Fixes**
   By identifying and fixing defects in the development cycle rather than after production, you can lower the cost of the fix. This approach yields:
   a. **Cost savings when contractors finish their work sooner than scheduled.**
   b. **Productivity gains from fixing defects in development.**
   c. **Cost avoidance by reducing the time and effort required to fix defects before production, since at least one study suggests that fixing defects in the development cycle costs 10 times less than post-production fixes.**

2. **Decreasing Help Desk Call Volume and Brand Erosion**
   Fixing defects post-production can lead to unnecessary help desk calls and compromised customer credentials. By identifying and fixing defects proactively, you can reduce help desk call volume and password reset infrastructure usage, minimize the risk of brand erosion from insecure websites, and protect your customers' sensitive information.

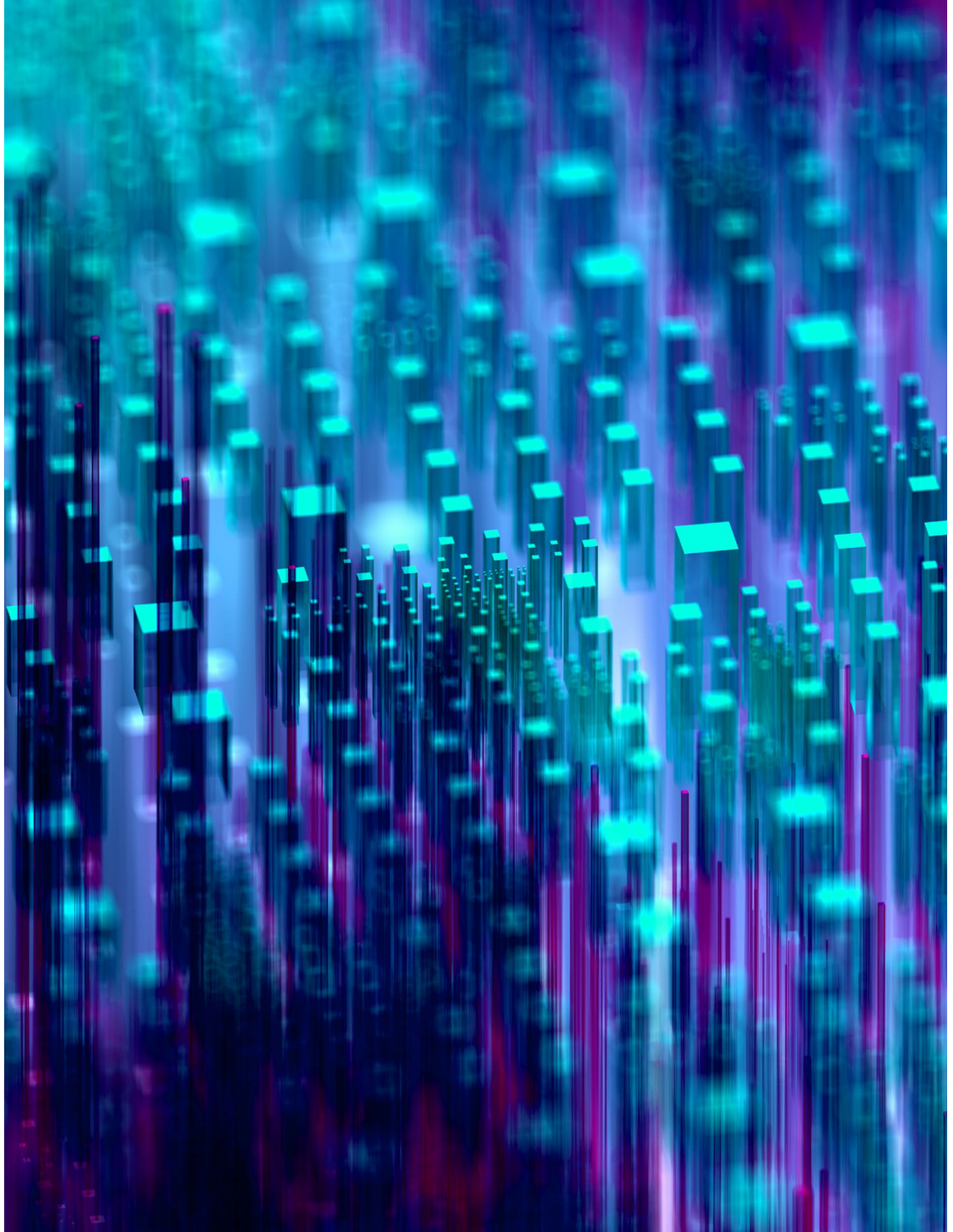
3. **Lowering Support Costs through Higher Quality Software**
   The need for costly maintenance and support decreases as software quality improves. By lowering code defect density, you can avoid a percentage of annual maintenance costs year after year, freeing up resources to focus on innovation and growth. This approach not only saves money but also ensures that your customers receive higher-quality software that meets their evolving needs.

Prioritizing the Pillars

When presenting your case to C-suite leaders, focus on the most attractive category: fixing defects in the development process. This approach offers the most tangible and immediate benefits, making it a compelling reason to invest in software security.

The second category, reducing help desk call volume and brand erosion, is a clear benefit, but may not be as persuasive as the first reason. However, it's still important, as improved security can protect your customers' sensitive information and maintain their trust.

The third category, lowering support costs through higher-quality software, has a longer-term impact. As software quality improves, the need for costly maintenance and support decreases, freeing up resources and reducing labor costs. In time, this approach can lead to significant cost savings, making it an attractive investment for your C-Suite leaders.



Business Case Preparation

To prepare this business case, you can gather the necessary information from your company's financial records. If you work for a public company, the balance sheet is usually under the "Investor Relations" or "Financials" section on the company website. This information is also included in the annual Form 10-K filing.

If you work for a private company, ask your CFO for a recent balance sheet. Once you have the balance sheet, estimate the annual costs associated with developing new software and maintaining existing software. This aggregated dollar amount will serve as the basis for your business case.

In some cases, the annual cost of software development and support can be substantial, measured in billions of dollars. In other cases, it may be in the tens or hundreds of millions of dollars. Review previous years to determine if these costs are trending up, down, or stable. This will help you build a solid business case and demonstrate your software security program's potential return on investment.

Calculating Average Unit Cost

Now that you have the total annual cost for software development and support, it's time to calculate the average unit cost, which, in this case, is the average hourly rate for developers. To do this, you'll need to estimate the loaded cost for employees and contractors.

For employees, you can use the actual average hourly rate plus 30% to cover the benefits. For contractors, you can use the actual rate plus 15% to cover the costs of providing infrastructure support, such as access to the code repository, IDE, VPN service, and other necessary tools.

Let's say the actual hourly rate is as follows:

Employees: $185

Off-shore contractors: $45

After applying the loaded cost, we get:

Employee rate: $185 + 30% = $240.50

Contractor rate: $45 + 15% = $51.75

To simplify the calculation, you can estimate the percentage of developers who work in each capacity. For example, if your company is approximately 60% employees and 40% contractors, you can work out a general combined rate by multiplying the rate for each group by the percentage.

In this case, the combined rate would be: $240.50 × 0.6 + $51.75 × 0.4 = $164.70

Determining Time to Fix

The next level of your model requires a calculation of how much time is required to fix software defects, which can be categorized into three levels of complexity:

- **Simple**:These are defects that can be fixed quickly and easily, often requiring minimal code changes.
- **Moderate**: These defects require more significant code changes and may involve some testing and verification.
- **Complex**: These defects are the most difficult to fix and may require extensive code changes, thorough testing, and verification.

To estimate the time required for fixing software defects, you can use measures that already exist or apply the following assumptions:

- Average time to fix a **simple** defect: **1-2 hours**
- Average time to fix a **moderate** defect: **4-6 hours**
- Average time to fix a **complex** defect: **8-12 hours**

These estimates can be used to calculate the total time required for fixing all defects, considering the number of working days and the average general hourly rate for your developers, which you calculated in the previous step.

Creating a Bottom-Up Model

The next step is to build a bottom-up model that will give you a more accurate estimate of annual savings. This will involve calculating the cost savings from reducing the number of defects, the time required to fix them, and any productivity gains from increased throughput.

I assumed that 50% of the high and moderate-severity security defects needed to be fixed. You can arrive at an accurate percentage by talking to your software development leaders and reaching a consensus on what you believe to be a reasonable percentage. Let's say the total number of high and moderate risk defects in a year was 100, with 50 high risk and 50 moderate risk. I chose the tools to be implemented during the development cycle that would identify security defects for the development teams and defined the labor cost necessary to implement the tools and support. I then calculated how many defect types were likely to be identified and remediated before production based on the current tools. I chose 30% of the high and moderate severity defects that would get fixed in the development cycle. The remaining 70% would be fixed after the code was released into production. Share these assumptions with your IT colleagues to align on what is reasonable based on current software engineering practices. You may select 70% fixed during development and 30% fixed post-production. It is more important to achieve a consensus with your development stakeholders versus being precise in the calculation. Once you are aligned, the fun begins!

Categorizing Software Defects

As you build your model, you'll need to make some key decisions about the types of defects to include. For example, in my business case, I only included labor to fix a combination of high- and moderate-risk defects. Less severe defects were not part of the business case. You can choose to isolate high-risk defects or combine them with defects that are moderate in risk. The tools you choose should be the source of the risk vulnerability.

The complexity of defects is already accounted for in the time estimates you provided earlier. You can make assumptions about the number of defects that will be complex versus those with moderate difficulty to fix. For example, you might assume that 20% of defects will be complex, 40% moderate, and 40% simple.

Your model must also determine the percentage of known defects that will be fixed in the software produced by your enterprise, both before and after production. I chose 50% in most cases, representing the number of known defects that will be fixed. Some of you may choose to go higher or lower. The key is to gain consensus among your IT leaders on a reasonable assumption.

You can choose to split the defects into security defects or not. If you do decide to split them, you'll need to estimate the percentage of security defects to be fixed. This can be a complex task, as the severity and complexity of security defects can vary widely.

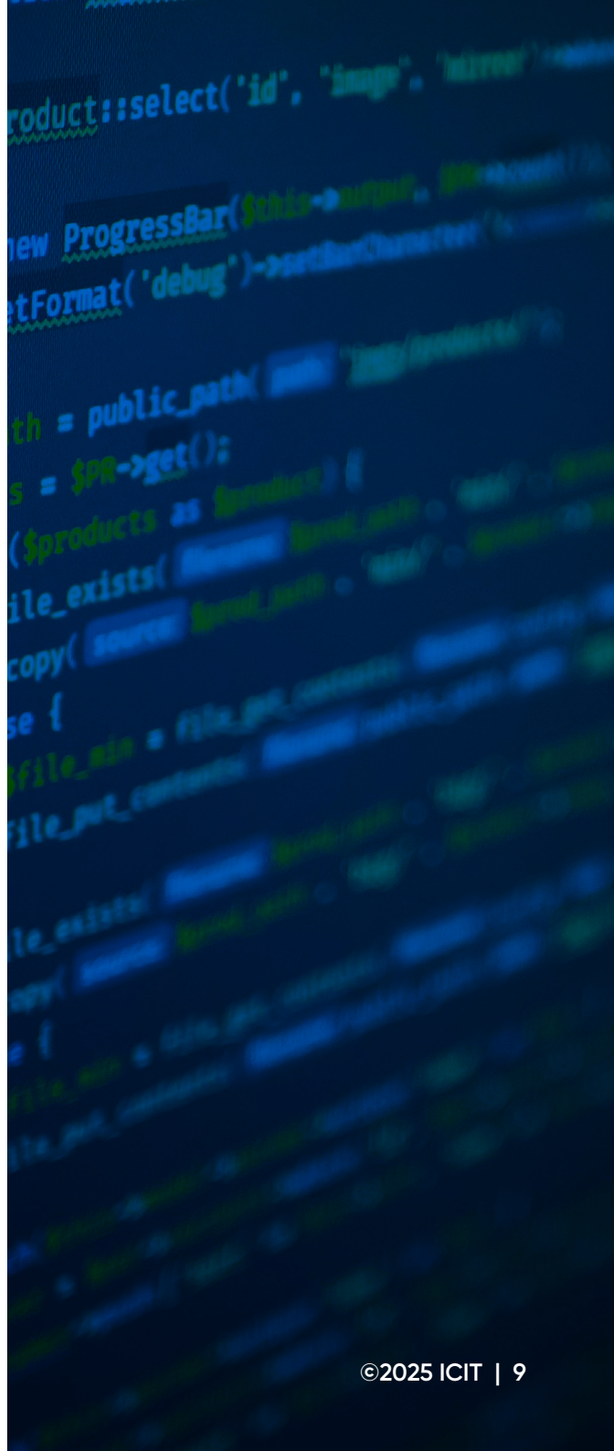
To better understand the defects that are usually fixed in your enterprise, I recommend asking your IT colleagues how many defects are fixed before production and how many are fixed post-production. If they measure this at all, they may prefer to identify the severity of defects to be fixed. For example, most enterprises live with defects in live software until the software is replaced. Significant defects that cause business impact typically get fixed earlier than less significant defects that have a minor business impact.

Whatever the percentage of defects fixed in your enterprise, it will offer a baseline measure useful in calculating the percentage you want to use in your model. This will help you make a more accurate estimate of the time and labor required to fix defects and, ultimately, the cost savings that can be achieved by fixing defects before production.

Here's an example of how you might structure this decision:

- **80%** of defects are fixed **before production** (high- and moderate-risk only)
- **20%** of defects are fixed **post-production**
- **50%** of defects are **security-related** and will be fixed before production
- **30%** of defects are **complex** and will take 8 hours or more to fix
- **20%** of defects are **moderate** and will take 4-6 hours to fix
- **30%** of defects are **simple** and will take 1-2 hours to fix

I assumed that developers would fix as many security defects as non-security defects. In this case, the total number of defects fixed would be 50: 25 high and 25 moderate. In terms of complexity, 15 will be complex, 10 will be moderate, and 15 will be simple, meaning the average time to fix a defect will be about 4 hours.



Fixing Defects in Production vs. Post-Production

The next step is determining how many defects can be fixed before production vs. after production. This is based on the tools selected and implemented to identify defects within the software development lift cycle, giving developers enough time to fix defects before production. In this scenario, TABLE 3 shows 15 defects that can be fixed before production, leaving 35 to be fixed after production.

Capers Jones made substantial contributions to measuring the output of software developers using either lines of code or function points. He also taught us that fixing software defects before launching code into production costs ten times less than remediation after the code is in production [2]. That's a tenfold savings of time that represents a benefit to the enterprise based on identifying the defects during the development cycle. That translates into almost $100K saved for the 15 defects identified before production.

At scale, saving developer time results in substantial savings to the enterprise, as seen in TABLE 2. The more effective your tools are at identifying software defects during development, the higher the productivity gain as your employees can use the time saved to do other project work. Contractors can be terminated when they complete their work, and the enterprise can apply those savings to the development budget.

TABLE 2: Cost Savings by Fixing Defects Before Production

	# of Defects	% of Defects	Hours of Work	Cost
Before Production	15	30%	60	$9,882.00
After Production	35	70%	1,400	$230,580

Calculating the Volume of Savings

As you refine your model, you can adjust the total percentage of defects fixed in addition to the percentage of defects fixed before production. This model is based on fixing 50 defects out of 100, but your enterprise's ultimate volume of code developed and defects produced annually will be significantly higher.

Consider the typical volume of code developed annually by your organization. Is it 100, 1000, or even 10,000 times more than 100 defects? As the size of the organization increases, so do the numbers. It's not uncommon for large enterprises to assemble millions of lines or tens of millions of lines of code annually.

The potential savings from fixing defects before production are substantial, and saving $4M annually is certainly a compelling argument for the C-Suite. As previously pointed out, capturing the productivity gain can be challenging, but even a productivity gain of 10-20% can look impressive, especially when considering tens of millions in annual spending on development efforts.

Learn From Others' Experiences

I took a thoughtful approach to building the business case in each of the six enterprises I worked with. Here are some key takeaways:

1. **Start with the facts**: Gather current facts about the software portfolio and the development/support effort.
2. **Engage key stakeholders**: Solicit input from the CIO and development leaders on the information they use for budget planning purposes.
3. **Get CFO buy-in**: Ask the CFO for help preparing the analysis and sharing a perspective on the business case.
4. **Seek consensus**: Aim for consensus among key stakeholders rather than a single point of view.
5. **Focus on the business case**: Emphasize the compelling aspects of the business case, such as detecting a high percentage of defects before production.
6. **Avoid a large SSG**: Consider relying on software security champions (5-10% of developers with security knowledge) rather than hiring a large software security support team.
7. **Champion automation**: Advocate for automation of the build process, which can reduce configuration defects and strengthen the business case.

By taking a thoughtful, data-driven approach, you can build a compelling business case for software defect prevention by reducing costs, improving productivity, and enhancing software quality.

Leverage the Assembly Line Analogy

The change in software development that delivers the greatest benefit to the enterprise is identifying and fixing defects during the development cycle rather than after the code is in production. When explaining the benefits of identifying and fixing code defects during the development cycle to the C-Suite, consider using the analogy of an automobile assembly line. Imagine an assembly line where different vehicle components are added to the car chassis. If 10% of the finished vehicles roll off the assembly line with dents in the front passenger door, there are two ways of handling the defects.

The first approach is hiring staff to fix the dents, repaint and buff the doors, and then send the vehicles to dealerships. This approach adds significant costs to the manufacturer, driving up the unit cost of production and increasing annual production costs.

The second approach is to diagnose the root cause of the defect, perhaps a front door assembly robot, and adjust the machine to eliminate the source of the dents. This approach may require a one-time investment in robotics maintenance or adjustment, but it eliminates the need for continuous repairs, which can be costly and resource intensive.

When developers assemble software components, it's like an assembly line. If defects are identified and fixed after the software is in production, it's equivalent to fixing dents in cars after they're assembled, adding significant costs to the production process.

On the other hand, identifying software security defects early in the development process is like adjusting the machine responsible for making the dents. By fixing software defects during the development cycle, developers can lower the unit cost of software development and reduce the risk of costly rework and defects in production.

Identifying Software Security Defects

There are several ways for developers to identify software security defects in the development lifecycle, including:

1. **Static application security testing (SAST)**: These tools analyze the source code for potential security vulnerabilities and issues without executing the code.
2. **Dynamic application security testing (DAST)**: These tools simulate real-world attacks on the application to identify vulnerabilities and weaknesses.
3. **Software composition analysis (SCA)**: These tools analyze the dependencies and libraries used in the application to identify potential security risks.
4. **Application security posture management (ASPM)**: These tools provide a centralized view of the application's security posture, including vulnerabilities, threats, and compliance issues.
5. **Software supply chain security (SSCS)**: These tools analyze the software supply chain to identify potential security risks and vulnerabilities.

Using these tools and techniques, developers can identify and fix software security defects early in the development process, reducing the risk of costly rework in production and lowering the unit cost of software development. Security defects identified in development often get fixed throughout the application, since defects are often replicated. Eliminating the defect earlier in the lifecycle may reduce the number of the repeated defects in the code base.

Spotlight on Defect Density

One of the primary advantages to measuring the software quality is to encourage the developers to improve, consistently resulting in lower support costs. A great way to enforce accountability for software resilience is to apply a defect density key performance indicator for every development area. Defect density is a measure that can be applied to any development group or business area. Here is an example in TABLE 3:

The development team working on the third software portfolio has the lowest defect density (1.50) compared with the other development teams. This means the code quality and resilience are higher in their portfolio. However, many factors affect the defect density score. Developers learning new technology products will have higher defects. Different languages, such as Go vs. JavaScript, will result in variations in defect density. The choice of tools used to identify defects is another variable. Despite these challenges, **the defect density score represents the most effective way to push accountability for resilience software where it belongs**: with the developers. The category of tools that perform application security posture management can measure defect density across platforms if the enterprise sets them up to do so.

I encourage you to ask your vendors if they can produce defect density scores for you. This gives your program an easy way to measure progress over time and across software portfolios.

TABLE 4 illustrates the likely progress when tracking defect density over time. It is very common for development teams to significantly increase their resilience, as measured by defect density, in very little time. It works across technologies and organizations. Sharing the results with three levels of the organization (developers, project managers, and C-Suite) provides the "ground truth" on software resilience and quality, driving the right behavior. Cybersecurity leaders do not have to be the driver. Sharing the facts at multiple organizational levels will drive positive change over time. Providing a consistent measure that is easy to apply, like the defect density score, is essential to this evolution. It offers a great way to demonstrate the business case that you provided to your stakeholders.

TABLE 3: Defect Density in Practice

	Lines of Code Produced	# of High and Moderate Defects	Defect Density Score
Software Portfolio 1	10,000	402	4.02
Software Portfolio 2	50,000	1,220	2.44
Software Portfolio 3	100,000	1,502	1.5
Average			2.65

TABLE 4: Measuring Defect Density Progress

Month	Lines of Code Produced	# of High and Moderate Defects	Defect Density Score
January	10,000	675	6.75
February	48,000	2,220	4.63
March	100,000	1,734	1.73
First Quarter Average			4.37

Additional Considerations

If I were preparing a business case for software security today, I would look carefully at ways to improve the instrumentation during the software development lifecycle to make developers accountable for creating for quality software. The maturation of product security professionals who understand the enterprise's software development and build process and work closely with developers is a positive change and represents an organizational model to be considered. In this model, defect density is an essential tool to measure quality and resilience.

I'd also design program components to include controls for using large language models (LLMs) to produce quality code. The current benchmarks for LLMs creating software represent progress but must continue to improve and reduce defects. The use of multiple models has demonstrated progress in quality. Still, the industry is not ready to accept code produced by LLMs without applying the same quality controls necessary for developers. Thankfully, the use of LLM models in tools to assess quality is improving rapidly.

Explaining to the C-suite that your software security program will reduce the annual cost of software development by 10% and the annual cost of software support by 10% represents a conservative estimate of the overall benefit. If your enterprise annually spends $100 million on software development and $500 million on maintenance, then the contribution from your program will be $6 million in a combination of cost avoidance and productivity. You can then consider the implementation costs, which are one-time costs spread over three years. Year 1 may enable a modest amount for cost avoidance, Year 2 higher, and Year 3 the full amount.

Implementation costs typically will include:

1. Software license costs for necessary tools.
2. Implementation costs for developer education and support of the champion model.
3. Operating cost for running the technical infrastructure to support the tools.

Conclusion

This approach to creating a compelling business case emphasizes the business benefits of a software security program and places less significance on risk mitigation benefits. Improving cybersecurity maturity for the enterprise is a clear benefit but measuring the probability of software resilience is less interesting to the C-Suite compared to lowering operating costs. Therefore, focus your business case on the latter and welcome the former when achieved. Getting top-down support for changing techniques and practices in software development is essential to supporting a multi-year implementation plan. The key to getting the support of your stakeholders is to deliver economic benefits that are easy for the C-suite to recognize.

Messaging to the Board

Cybersecurity leaders should provide the ammunition necessary for the CEO to share the program's business case with the board of directors. In a public company, this is the primary responsibility of the CEO. A CISO should equip the CEO with a story illustrating the economic benefits of building software right the first time. CISOs should also be prepared to offer more descriptions of the software security program if asked to do so in a board meeting. Board members don't approve software security programs, but they certainly are incentivized to understand the business case for the investment. In a private enterprise, it is more likely that the burden of describing the benefits will be on the CISO as they support the CEO's messaging and answer questions from board members. Using a story, like the auto manufacturer, is the most effective way of sharing the rationale for this investment.

Reference List

1. G. McGraw, J. Viega, and G. Hoglund, *Software Security Library*. Upper Saddle River, N.J: Addison-Wesley, 2006.
2. C. Jones, *Applied Software Measurement*. New York, USA: McGraw-Hill Professional Publishing, 2008.

Copyright

The Institute for Critical Infrastructure Technology

The Institute for Critical Infrastructure Technology (ICIT) is a nonprofit, nonpartisan, 501(c)3 think tank whose mission is modernizing, securing, and making resilient critical infrastructure that provides for people's foundational needs. ICIT takes no institutional positions on policy matters. Rather than advocate, ICIT is dedicated to being a resource for the organizations and communities that share our mission.

Founded in late 2014, the Institute's work has earned the trust and respect of the nation's most influential institutions and serves a diverse community of technology, policy, and business leaders. By applying a people-centric lens to critical infrastructure research and decision-making, our work ensures that modernization and security investments have both a lasting and a positive impact on society.

Jim Routh

A board member, advisor and investor with specific expertise as a transformational security leader focused on applying risk management discipline to a converged security function for global enterprises to achieve enterprise resilience. Demonstrated track record of designing security control using innovation and data science to align senior executives to deliver world-class level security capabilities to drive positive business results in a digital world.

CyberRisk Alliance

CyberRisk Alliance provides business intelligence that helps the cybersecurity ecosystem connect, share knowledge, accelerate careers, and make smarter and faster decisions. Through its trusted information brands, network of experts, and innovative events it provides cybersecurity professionals with actionable insights and act as a powerful extension of cybersecurity marketing teams. CyberRisk Alliance brands include SC Media, the Official Cybersecurity Summits, TechExpo Top Secret, InfoSec World, Identiverse, Cybersecurity Collaboration Forum, Cybersecurity Collaborative, Security Weekly, ChannelE2E, MSSP Alert, and LaunchTech Communications.

ICIT

CyberRisk ALLIANCE