Software Concepts and Issues for Senior Leaders

A video series featuring SEI Fellow John Foreman

Video 1 Script

DM19-0686

HOST: Hello, and welcome to the first in our series of videos on Software Concepts and Issues for Senior Leaders. I'm [name, title] and I'm here with SEI Fellow John Foreman who will share with us his knowledge, experience, and insights about the latest issues, opportunities, and selected topics related to software technology, cybersecurity, and software acquisition management. The purpose of this series of videos is to provide senior leaders who oversee system acquisition the skills and knowledge to

[PRODUCTION: DISPLAY BULLET POINTS ON SCREEN OVER LIVE SHOT, PREFERABLY AS HOST READS THEM]

- review programs at key milestones
- determine programs are executable and on-track
- proactively recognize the symptoms of key problems
- implement recovery strategies

Welcome, John. Can you tell us a little about your background and your history at the SEI?

JF: Certainly and Thank you, Julie.

I was very fortunate to be able to build and sustain software systems, in particular C3 and Intel apps, in the Air Force. This was at a time when the AF was transitioning out of organic development, except for some special category organizations. After my development time, I was honored to serve on the Computer Science faculty at the Air Force Academy. After my AF service, I spent 6 years at Texas instruments, working software technology and Ada compilers and run time systems. At the SEI, I've largely been focused on direct customer challenges, many red teams and transitioning new ideas and concepts to the customer community. I also had the good fortune to spend 4 years at ARPA directing some software technology programs. Lastly, I've developed training materials and "coaching" sessions on software technology and issues

which has been provided to leadership in DoD organizations. Our management here in the SEI's Software Solutions Division thought these topics would be of interest to leaders and executives in a variety of organizational settings. So, they asked me to record this series of videos, which the SEI will make freely available.

HOST: Thank you, John. I understand you will devote the first two videos in this series to laying a foundation for the topics to follow.

JF: That's correct. I want to begin today by providing a brief introduction to the SEI for those who might not know about us. (Our website provides a wealth of information about our history and mission.) I also want to discuss some software fundamentals and explain why they're so vitally important. I'll conclude with some software "horror stories," some of which have been reported in the news and some that derive from independent technical assessments the SEI has conducted. After we get thru the foundational materials, the plan is to produce focused videos on relevant topics such as project management, agile methods, DevOps, various aspects of cyber security and software and info assurance, acquisition and life cycle management of software systems, system reliability, software tooling, integration and test, artificial intelligence, etc.

HOST: That sounds great. Why don't we dive in?

JF: First a little background on the SEI. The SEI is a federally funded research and development center, the only such center focused on software. Our primary mission is to support the defense of the United States; indeed make software a strategic advantage for the department of defense. We conduct research on software engineering, systems engineering, cybersecurity, and many other areas of computing, and we work to introduce private-sector innovations into government. We also work with the private sector and academia in an array of disciplines.

It's not unusual for DoD customers to seek our help when they're faced with a technically complex software issue impacting system or program performance issue, or when a fix is needed immediately, or vital cybersecurity challenges must be overcome

Customers also seek our help for extended engagements on more systemic software challenges and where we perform analyses to inform customer decisions about system and enterprise solutions and software capabilities.

HOST: That's great, John. And I would direct our audience to our website at SEI dot CMU dot EDU for a complete picture of all the good work we do here. You also wanted to discuss software fundamentals?

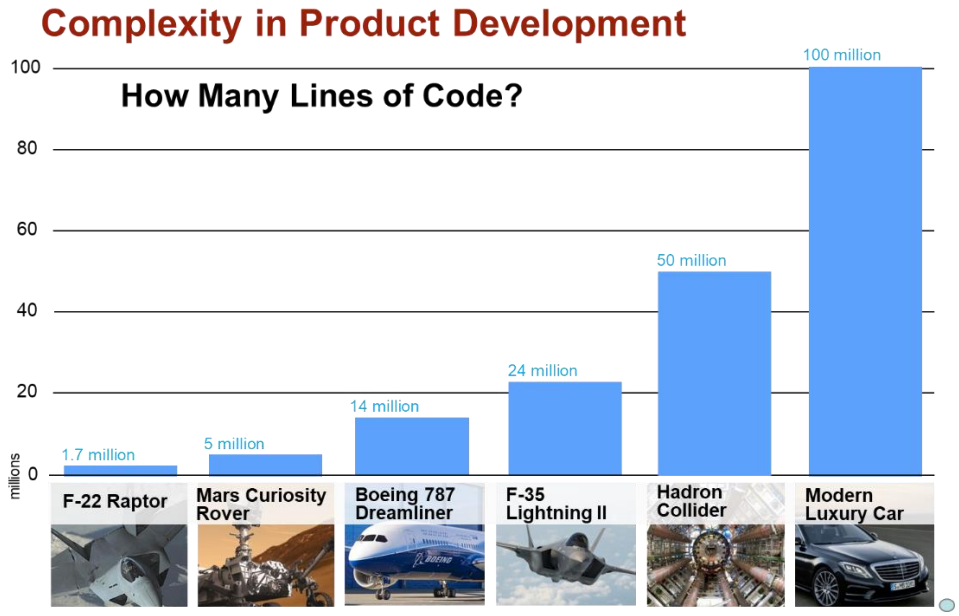
JF: Right.

I think it's fair to say that software has become pervasive in all aspects of life – just think medical devices, your cellphone apps, and enhanced safety systems on your automobile. Similarly, software is essential to a vast range of military systems capabilities and operations. Indeed, greater than 80%+ of some weapon system's functionality is software dependent!

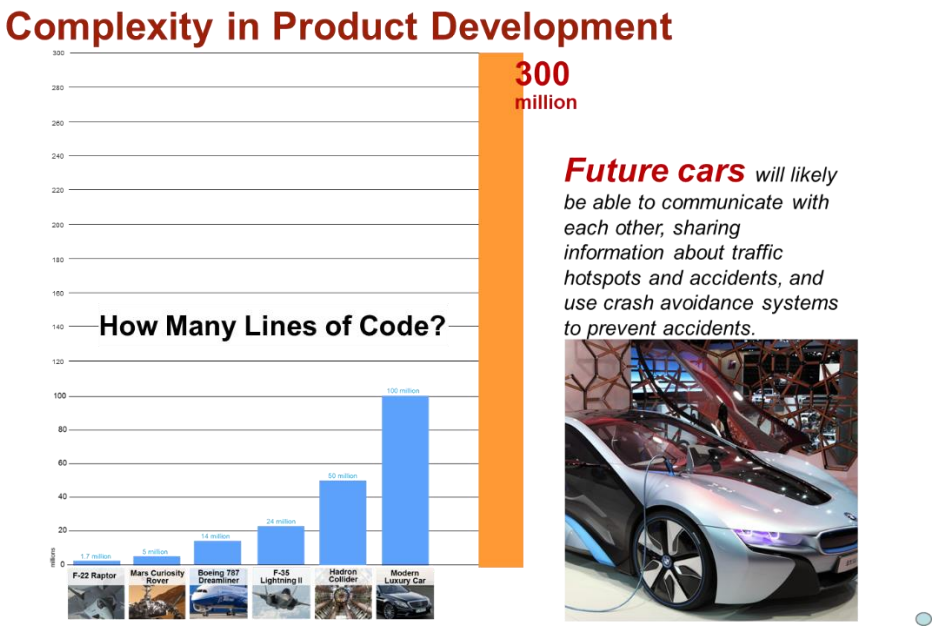
[PRODUCTION: SHOW COMPLEXITY in PRODUCT DEV SLIDE/CHART]

As shown on the chart, millions of lines of code go into modern systems, further emphasizing the importance of software in these products, and its role in delivering new capability. While some may doubt the # of lines of code in a modern luxury car, consider that much of the software is componentized into sub systems and just the Anti-lock braking systems (ABS) have something like 3M LOC by themselves.

Some projections for future cars, which will be able to communicate with each other, run autonomously, share info about traffic hotspots and accidents, and provide crash avoidance systems run to 300 Million LOC



[PRODUCTION: SHOW SECOND COMPLEXITY SLIDE]



To attempt to put all these #s in some perspective, a million lines of code, printed, would amount to 18 thousand one-side pages, which stacked would rise six feet high!!! Imagine trying to comprehend that! Typically, we find that even as the lines of code in a product grow, the production schedule for these projects often gets shorter. And because the typical rate of code defects is 5 to 10 defects per thousand lines of code, a product with a million lines of code can ship with 5 thousand to 10 thousand defects. Thus the absolute necessity for early test and integration cycles.

DoD acquisition programs often ignore these realities. This results in unrealistic schedules and unplanned test-and-fix cycles inserted to try to make the software more reliable.

HOST: So, all this capability comes with a price, and things aren't getting any simpler?

JF: Correct. And that's not the only challenge software poses. There are a number of challenges leadership often fails to consider, for instance…

[PRODUCTION: SHIFT THROUGH A DISPLAY OF THE BOLDED BULLET ITEMS AS JF DISCUSES THEM.]

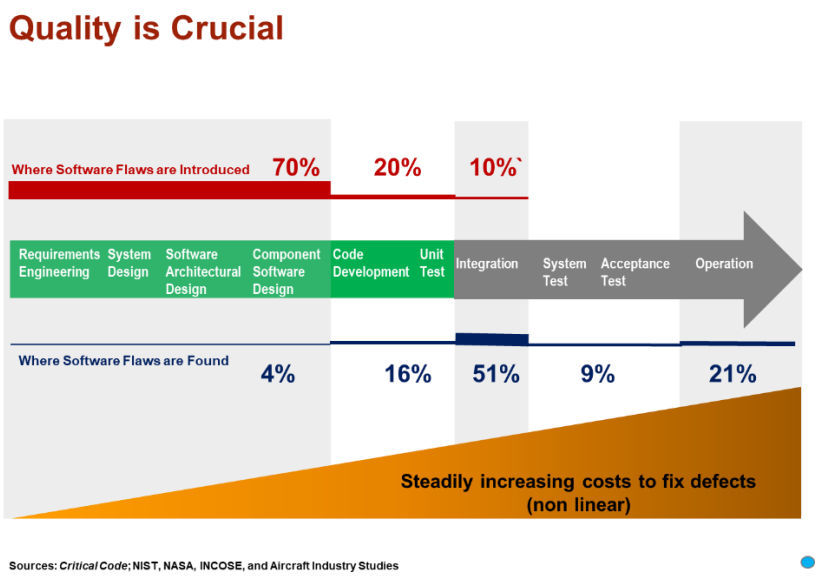
- **Software is mutable**
  - This is both good news and bad news.
  - Limited concept of standard parts; almost everything is custom development
  - Complexity management is an ongoing challenge
- **Another challenge is that software tends to be invisible**

    - Typically we don't buy code – we buy a *system*.
    - Delivered code is but one part – support/testing software can increase total by 10 to 100x.
- **Leaders also need to understand that doing it "right" requires persistence and discipline.**
    - These are not often the most common human traits.
    - Adopting good software engineering practices has good ROI, but "cash-flow" problem
- **It's also the case that few key decision-makers have in-depth software understanding.**
    - Software management and engineering competency, while improving, is generally weak.
- Finally, **software is the major schedule and cost driver for development and sustainment of most systems, embodying "unlimited" complexity.**
    - Software and hardware technology continues to evolve very rapidly.

HOST: So, software has become a huge, complex, and growing component of DoD and other systems, but the leadership responsible for these systems hasn't always recognized these challenges?

JF: That's right, and all these challenges impact quality, which is crucial. And the earlier in the development lifecycle questions of quality can be addressed, the better the end product. What's more, fixing problems early in the lifecycle will reduce overall cost, and limit schedule growth. Let's have a look at this chart.

[PRODUCTION: DISPLAY "QUALITY IS CRUCIAL" CHART]



This chart is the reason SEI needs to assist programs especially at their inception

We also have to remember that software engineering depends on humans. Humans make mistakes. Undetected the mistakes result in defective software, cyber vulnerabilities, and systems that fall short of mission capability requirements.

What we know is that 70% of the flaws are injected *before* coding begins, and 90% of the flaws are injected *before* testing begins, but 80% of these flaws are only found and removed *after* code development is completed.

Late discovery and repair of these flaws increases development costs, causes significant schedule delays, incurs rework costs in excess of 50%, a 2- to 5-times increase in five-year TCO (total cost of ownership), and a 5- to 10-fold increased likelihood of high-severity critical failures. What's more, we are challenged by the fact that When coding is complete, the work may only be about half done.

So, to return to the chart, the highest-leverage technical practices occur generally to the left.

If you get the up-front engineering right, you can eliminate these problems.

Our technical focus at the SEI is on the left, including requirements, system design, architecture design, security engineering, component design, and introduction of agile methods

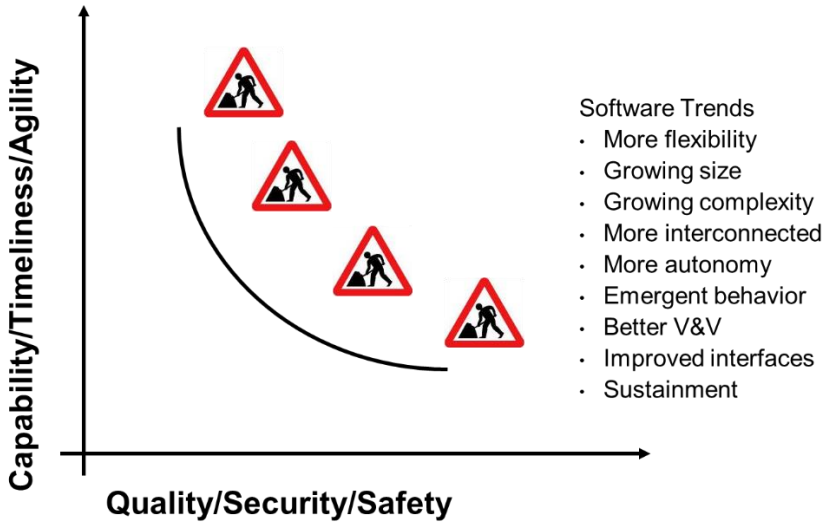
HOST: So, what I think you're saying is, "Bad news doesn't get better with time." In other words, leaders need to come up to speed on the challenges of software development and plan to address them at the earliest possible stages of the development lifecycle.

JF: That's correct. But the challenges don't stop there. For years, customers, warfighters, and user demands have been driving changes that continually expand the frontiers of software engineering.

[PRODUCTION: SHOW THE FIRST "FRONTIERS OF SOFTWARE ENGINEERING SLIDE]

If you look at this graphically, the trends are certainly toward increased capability, timeliness, and agility, as well as increased quality, security and safety

The Frontiers of Software Engineering



To get there we need systems with

- More flexibility
- More connectedness
- More autonomy
- Better more complete test coverage
- And better interfaces

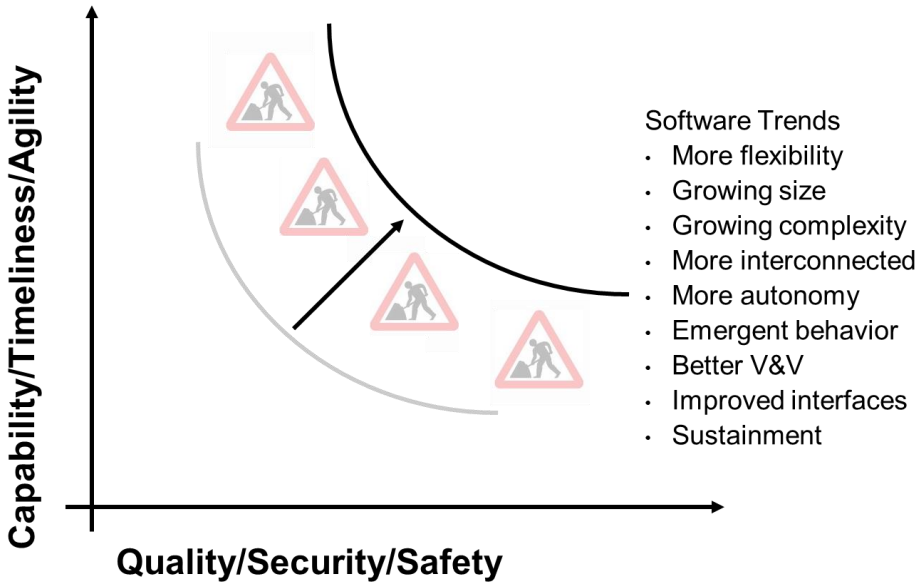
And users are demanding these improvements in environments that have traditionally been considered inhospitable to computing. This is a particular need of the DoD, which requires more computing capability at the tactical edge.

Software developers are also part of this equation. They need for ways to handle larger, more complex software programs, including improved technologies for validation and verification and sustainment in order to move further ahead, as shown in the next slide

[PRODUCTION: DISPLAY SECOND FRONTIERS OF SOFTWARE ENGINEERING SLIDE]

So, the software engineering research community continues to seek better techniques and methods in this ongoing struggle to increase system capability

The Frontiers of Software Engineering



Eventually these forces and the work ongoing will move the frontiers – so that the level of quality matches the level of capability.

[NOTE: I think when it comes to the "horror stories," which follow, John should adopt a more relaxed, anecdotal approach.]

HOST: So, these are all realities that leaders of software development programs need to account for. I understand you have some anecdotes of real-world scenarios you've encountered in your work that illustrate what can happen when you fail to consider these realities.

JF: Yes, I do. First I want to overview some items our viewers might have read about in the news. And what the news tells us is that software and security failures are rampant.

[PRODUCTION: IS IT POSSIBLE TO RECREATE HEADLINES FROM NEWS SITES IN A WAY THAT DOESN'T VIOLATE PERMISSIONS/FAIR USE? IF SO, WE'LL MAKE UP SLIDES THAT CORRESPOND TO JOHN's TALKING POINTS HERE AND DISPLAY.]

[The following are items from John's slides; perhaps cover two or three. Note that John may have others in mind for this part of the talk.]

- Toyota is recalling millions of Prius Hybrids to fix a software bug
- iPhone Security Flaws Exposed
- Software Error Shakes Bitcoin Exchange
- Microsoft Operating systems and Brower Flaws

- Data breaches at Target and other major retailers

In addition to items like these, it's important to note that a 2014 Poneman Institute study reported that 47 percent of U.S. adults had their personal information exposed by hackers.

HOST: It seems like there's a new report of major bug or security breach every week.

JF: It's certainly a problem. The other point about these failures is that they're expensive. *According to another Poneman Institute report, this one from 2013, the average cost of a security breach is 188 dollars per record. That adds up fast when you're a large commercial or government enterprise that's had millions of records exposed. ???*

HOST: So, software flaws can be large in scope and very costly. But it's not just a problem for the big organizations we read about in the news, correct? You have some hair-raising stories from your experiences working on independent technical assessments for the SEI. Care to tell us about a couple of those?

JF: Certainly. The first involved a satellite control system. The problem in this case stemmed from poorly documented requirements, poorly written test plans, and a very immature attitude toward software testing.

[PRODUCTION: SHOW PHOT [OR VIDEO] OF SATELLITE IN ORBIT]

The system testing only accounted for the "happy path." In other words, "Does the system do exactly what it is supposed to do, when it receives exactly the expected input?" Almost totally ignored were "atypical"/unexpected situations.

An operator was asked to submit invalid input into the system (in a test environment) – and the system lost the satellite and provided no mechanism to identify that an error had occurred.

Further ad-hoc testing revealed that this lack of robustness was pervasive throughout the system. There were multiple points at which incorrect user information could cause the software to lose contact with a satellite, without triggering an error message, creating significant risk in operational use.

The program office's position when defining requirements and testing was that users of the system were well-trained; thus, errors in system input were unlikely to occur. The system did not have requirements to be robust in the face of erroneous user input.

Defects associated with robustness tests were not logged when the system was rolled into production. Operational users were not made aware that the system would not perform error checking on user inputs.

However, the system was deemed "passed" because the poorly written test plans and poorly documented requirements were met, and the software was fielded despite these known critical deficiencies.

HOST: That is a pretty terrible outcome. I think that demonstrates how failing to apply good software engineering practices right from the start can have disastrous consequences down the line.

JF: That's right. In this case, testing was deferred until the end of the program when it certainly could have been made an incremental activity, and schedule was more important than quality

Another "horror story" that comes to mind has to do with an IT system focused on personnel management that fell significantly behind schedule. As opposed to specific testing issues, this program had more widespread problems.

First of all, the **requirements** were not clearly written or properly vetted. This caused significant rework and allowed for scope creep.

Then there were **design** issues. During development, functional users continued to request (or demand) additional features, without making (or being educated about) schedule concessions. This scope creep, as you can imagine, led to delays. For instance, the process for managing changes was very loose. The government team approved about 85% of the proposed design changes without analyzing cost or schedule impact.

Testing also led to delays. Sample data to test external interfaces was not made available for all systems prior to the start of system integration testing. This inhibited interoperability testing with legacy systems. What's more, the contractor and government test environments were dissimilar.

The lack of **metrics** was another significant issue in this case. There were no contractual requirements for reporting progress and quality metrics. No metrics were collected on known critical issues such as requirements volatility or defect density. *[Relate to schedule delays?]*

Finally, the **documentation and contract document requirements lists (CDRLs)** failed to require a final software deliverable - delivery of software documentation or completed final code. It wasn't in the CDRL list. The CDRLs also did not include any metrics on the progress of the activities or quality of the product. So, what did the government have to work with? What was the real code baseline and configuration of the system?

HOST: All this highlights the need for leadership knowledgeable in software development and software engineering, and the need to apply solid software engineering practices from start to finish.

JF: Yes, and the earlier in the process the better. Stories like these—and I have a lot of them—demonstrate why sound software engineering practices are needed, especially for systems critical to our national defense and for safety-critical systems.

So, the bottom line here is:

- Software has become one of the most important components of our Nation's weapons systems, and it continues to grow in importance.
- Software defines the way our systems see, communicate, and operate in combat.
- Design and acquisition decisions at the beginning of the software development process frequently have far-reaching and long-term effects that impact the weapon system's efficacy on the battlefield and its ability to adapt to changing requirements.
- Problems associated with software development continue to plague major DoD acquisition programs—resulting in long delays in fielding, significant cost overruns, and, program cancellation.

HOST: Thank you, John, for sharing your insights. As we noted, this is the first in a series you're preparing to help senior leaders understand software engineering concepts. What will talk about in your next installment?

JF: Next time I'm going to discuss the things every senior leader needs to know and do to enable their project to be successful. Then, in each subsequent video, I plan to take a bit of a deeper dive into specific topical areas.

HOST: That sounds great, John. I'm looking forward to it.

[We need some "outro" words here and, perhaps, a pointer to where these videos will live. We might consider doing this as a voiceover since this is still to be determined.]

[We might consider adding a slide at the end with credits for any sourced material.]