Software Sustainment of Mission-Critical Systems:

Wringing Value out of Legacy Assets

IEEE STC

March 29 – April 3, 2014

ASRC FEDERAL

Customer-Focused. Operationally Excellent.

Wringing Value out of Legacy Code

Legacy systems tempt both acquisition and development communities alike

- Mission owners (usually) appreciate their fielded systems
  - See the value from historical development
- Development teams see them as new markets and new capabilities
  - Significant up-front work has already been done
- Both parties can reach the conclusion that adding new missions to existing systems is a “Win/Win” situation

Savings provided by sustaining existing systems

- Infrastructure and foundational pieces are already written
- Labs, testing environments and facilities are often already stood up
- Certification and accreditation approaches are already vetted
- System already works

There is nothing left to do but highly visible and rewarding stuff!

The Economic Case For Sustainment

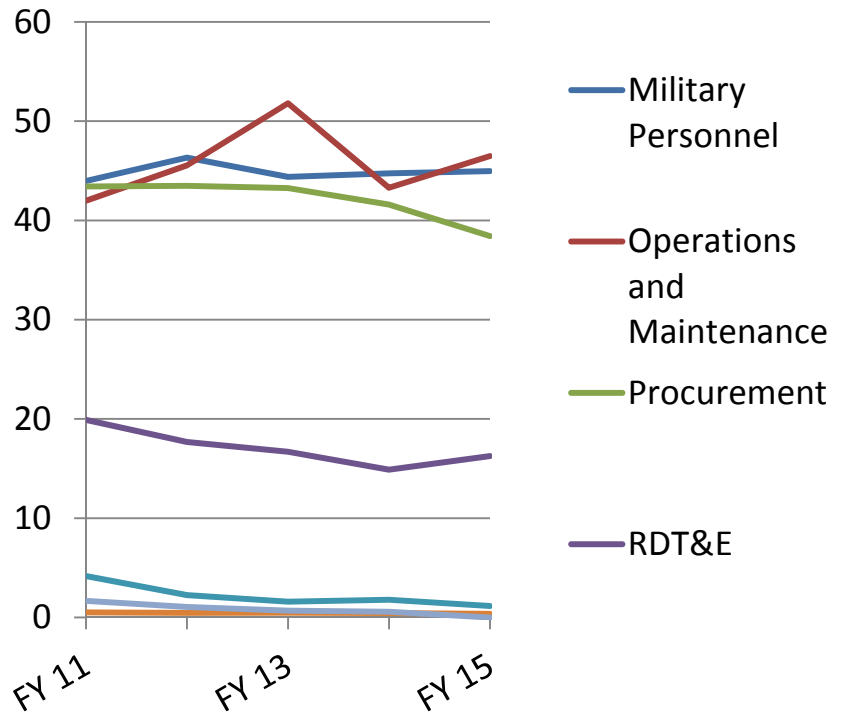
Very capable legacy systems are still vital, essential systems

- Support national defense, power transmission, and processing and manufacturing
- Risky systems to replace due to impact of any failures

Some systems will take decades to replace with modern ones

- May not be cost effective to replace all at once
- Numerous opportunities for a capable developer

Department of The Navy Budget by Category



O&M activities outpace historically flat RDT&E ones

Types of Software Sustainment

Software sustainment is not a one-size-fits-all activity*

- **Corrective Sustainment** – diagnosis and correction of program errors after its release.
- **Perfective Sustainment** – the addition of new capabilities and functionality to existing software.
- **Adaptive Sustainment** – modification of software to interface with a changing environment.
- **Preventive Sustainment** – modification of software to improve future maintainability or reliability.

Understanding which flavor(s) of sustainment will be performed is crucial to project success

**Program Manager's Guide for Managing Software*, 0.6, 29 June 2001, Chapter 12: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc as quoted in http://www.stsc.hill.af.mil/resources/tech_docs/gsam4/chap16.pdf. Orig. link broken

Sustainment is Far from Simple

Sustainment of legacy systems presents a developer with a unique set of challenges

- Software that wasn't organically developed, and not very well understood
  - May have limited-to-no visibility into what is running inside the legacy system
- Often written in an older language, and at a different place in time

There are significant potential "gotchas" in any sustainment phase

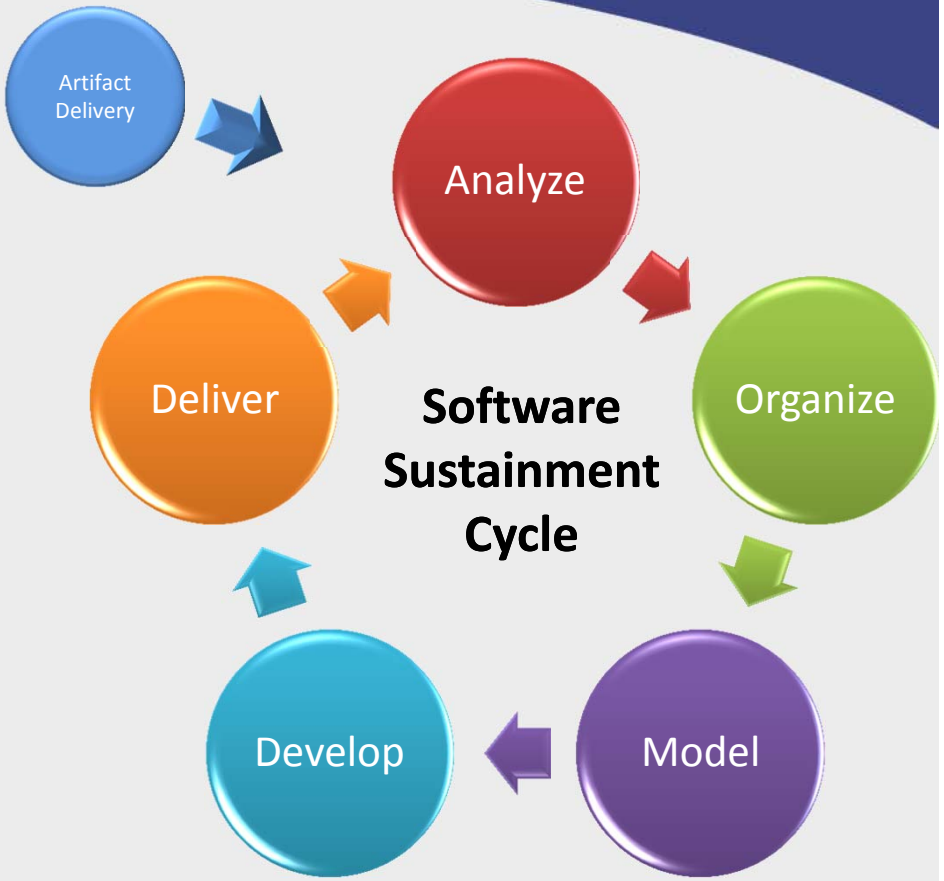
- Under-estimation of effort
- Addition of updated or additional requirements
- Limited availability of hardware, software and licenses

Software Sustainment Overview

There are five phases to our software sustainment approach

The start of the cycle depends on the origin of the software

- In a sustainment model, the cycle begins when initial artifacts are accounted for, such as
  - Requirements/Documentation
  - Source Code
  - Test Plans
  - IA, software assurance and security plans



Having a sustainment approach prior to actually performing the work is vital

Entering the Sustainment Cycle

ASRC FEDERAL

Artifact Delivery

Catalog all system artifacts

- This is often the most time-consuming phase
- Significant risk to cost and schedule if done incorrectly
- This process should begin prior to start of project

Identify any missing artifacts (and there will be missing artifacts)

- Documentation and models
- Source code and testing artifacts
- Plans and procedures, configuration management artifacts

Any documentation gaps need to be accounted for in the development plan

- They will likely affect cost and schedule
- Prioritize gap-filling efforts; not all artifacts are created equal

Analysis

Gather a fundamental understanding of the system

- Requirements and their traceability to existing components
- Organization of the system
  - What interfaces exist, and how are they used?
  - What file system organizational techniques were employed?
- How the system fulfilled the requirements that it was designed against
- Intent of the system's designers and developers

Software engineering tools aid in the understanding of both the software code and the executable content

- Integrated Development Environment tools, code profilers, component modeling approaches
- Great way to fill gaps in documentation and ground development efforts to reality
- Don't forget about performance requirements!

Analysis Tools

Analyze

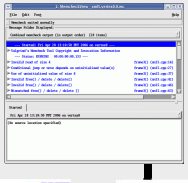
ASRC FEDERAL

Valgrind

Runtime tool suite to detect and debug memory management and software threading issues

NightTrace

Interactive debugging and performance analysis tool that displays kernel and application software activity

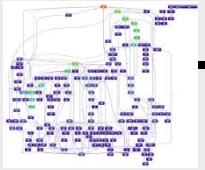


Insight

Toolsuite to assess operability of critical processes and interfaces in the computing environment, and validate the configuration of operating environment components

OProfile

Provides an understanding of software behavior by displaying execution frequency of software threads



Ntop

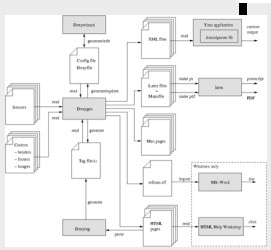
Network probe and GUI to display interactive network usage and status

Example Output:

FILE	LINE#	ERROR CATEGORY	ERROR TYPE
file1.cpp :	21	memory leak	: dynamic_buffer
file2.cpp :	110	resource leak	: file_descriptor
file10.cpp :	204 (error)	overlapping data buffer	: copy_destination

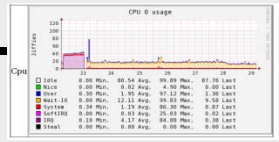
CPPCheck

Static analysis of source code for various software vulnerabilities, such as memory leak detection



Doxygen and other documentation tools

Automatically document source code for existing systems.

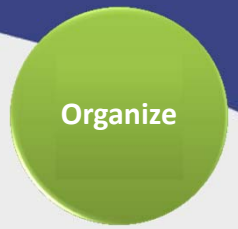


Collectd and DDA

Ability to monitor and analyze CPU, memory and network utilization at the system and component level, correlated with Data Recording Extraction Points

A full range of COTS, FOSS, and developed tools can assist analysis activities through automation

Organization



Optimizes the organization of the system to facilitate easier sustainment

- Focus is placed on maintainability and reproducibility, as opposed to rapid capability development
- Organization and structure of the software is oriented towards repeatable, incremental deliveries

Documentation artifacts are aligned with the software deliverables

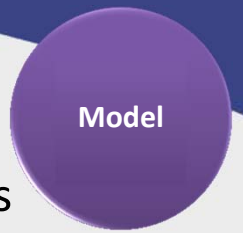
- Requirements and traceability
- Test plans and procedures
- Information assurance plans

Process tailoring occurs in this phase

- Ensures that processes align with expectations and deliverables

A plan to fill existing gaps is key to phase success.

Model



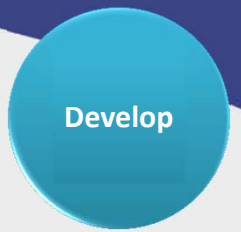
Modeling of software and systems artifacts greatly enhances productivity and reduces overall development costs

- There are a variety of modeling tools that can automate the process
- These tools provide a variety of modeling services in addition to model-based development
  - Reverse Engineering of existing source
  - System, component, and source code modeling
  - Database Schema modeling and development
  - Interface design and model-based interface management
  - Modeling reports and model-based documentation
  - Enable refactoring

These model-based artifacts become a part of the software deliverable process

- To support forward-, round-trip, and any subsequent reverse-engineering
- Can replace existing documentation artifacts and/or fill CDRL gaps

Develop



- Don't underestimate the need for legacy and specialty language experience and expertise
  - Ada, CMS-2, JOVIAL, and FORTRAN
  - Assembly language, and proprietary hardware-specific languages
  - C, C++, C#, .NET, Real-Time Java, J2SE, and J2EE
  - Analysis languages, including MatLab and LabView
- Maintain data on productivity and performance specific to sustainment
  - Modernization/development during sustainment will likely yield different data than your green field development process
  - Develop heuristics to guide development (the 25% rule)
- Track and respond to the unique sustainment metrics
  - Look to quality standards (e.g., CMMI and ISO) for metrics and quality guidance

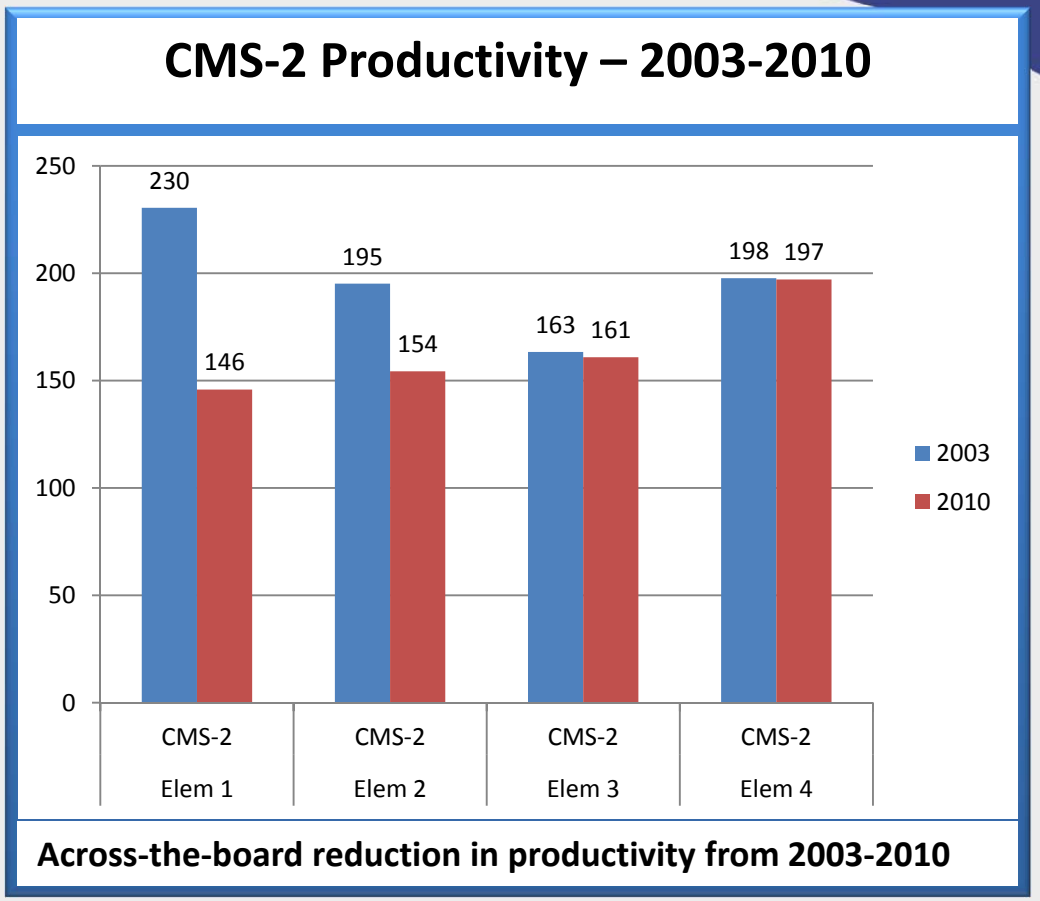
Remember, some types of developers are getting scarce!
Laws of supply and demand apply…

Costs of Performing Sustainment Work

Historical sustainment metrics show a decline in productivity using legacy languages

- Working population shifted to new technologies
- Decrease in available lab time and computing resources
- Heavy reliance on direct code and patching

Driving force in determining whether to modernize



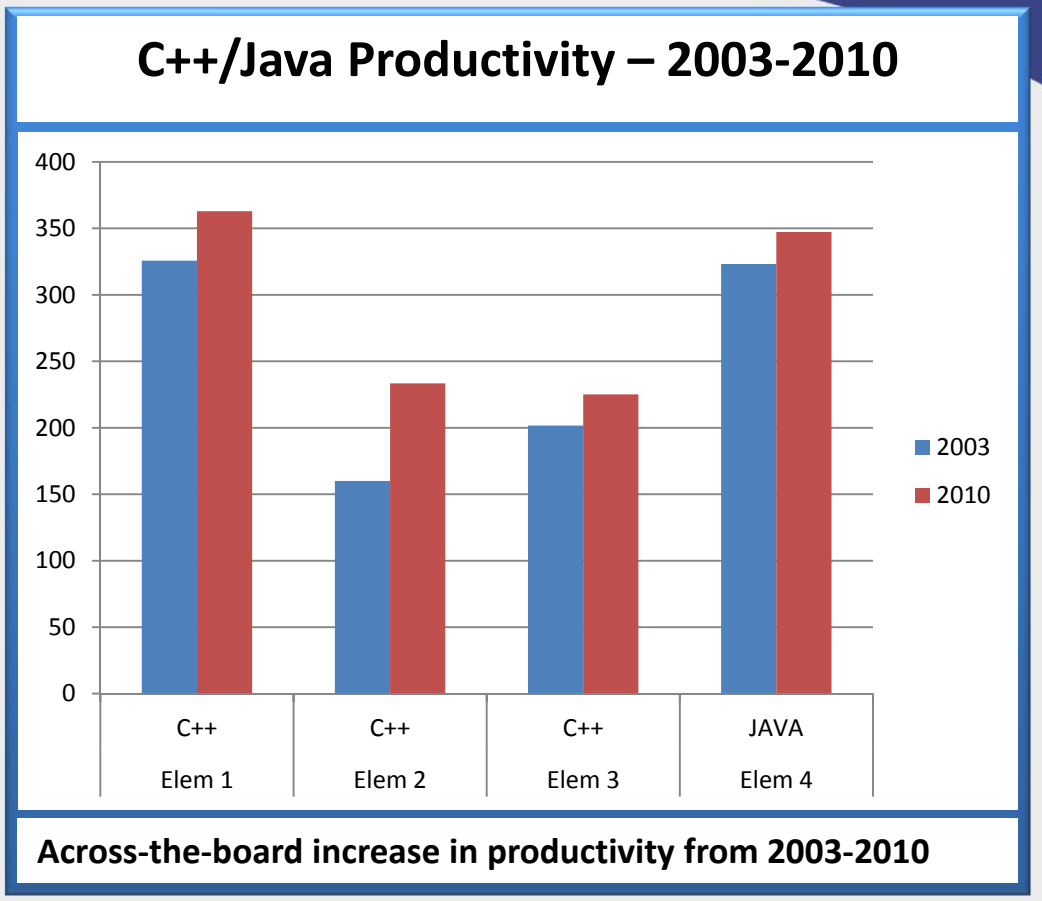
Legacy software developers and sustainers have relatively expensive skillsets

Increase in Modern Language Productivity

Modern language productivity increased even as legacy language productivity decreased

Modern language developers are more readily available and cost-effective

Reinforces the shift away from legacy languages



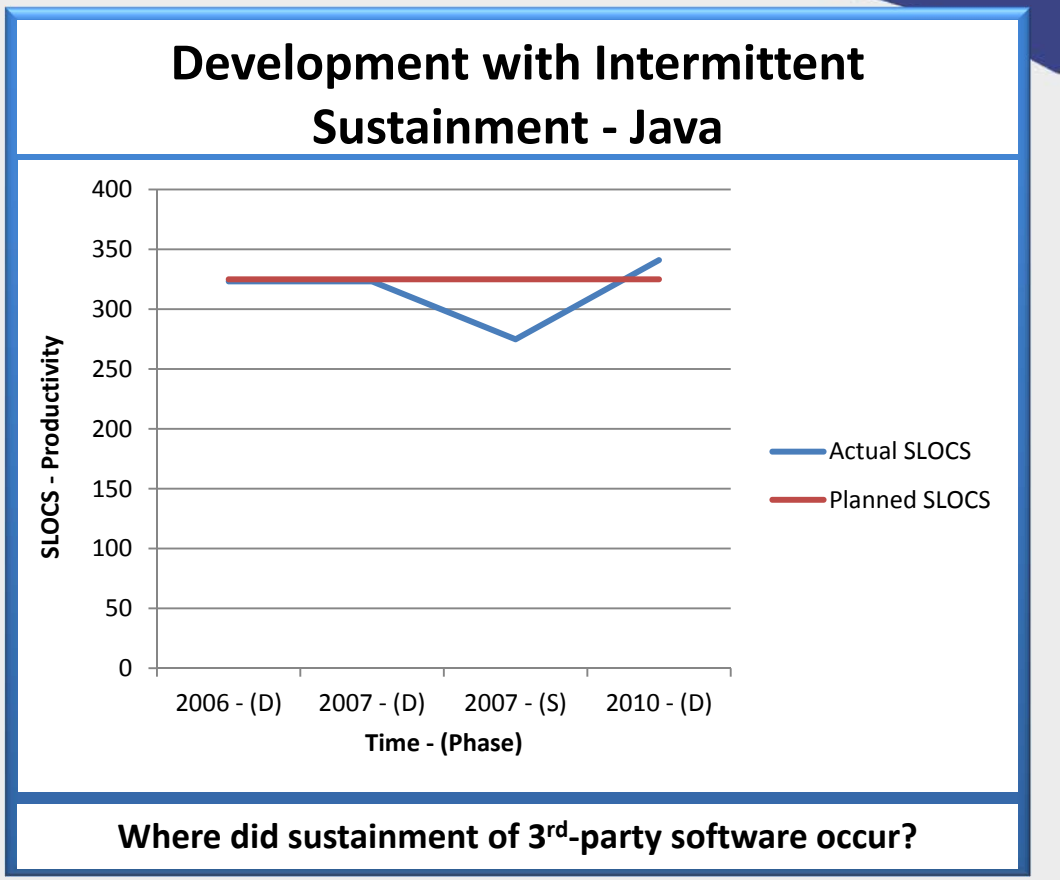
Modern language developers have higher productivity and are often more cost-effective

Planning and Pricing Sustainment Development

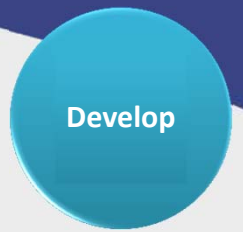
Accept that sustainment is a different discipline than development

- Development metrics may not be suitable for planning sustainment activities
- Consider the impact to the bottom line

Look before you leap, and record metrics along the way



Development Testing



- Historical test plans are a big aid to testing a newly sustained system
  - Foundation for development test plans
  - Now is a good time to modernize the approach
- Test results need to be evaluated differently in a sustainment model
  - Additional information needs to be acquired and maintained
    - May require a process change
  - Source of software defects needs to be determined and recorded
- Track and respond to testing metrics differently than you would during new development

Testing process may need to focus more on regression testing, and be more "manual" than development testing

Deliver

A variety of skill-sets is represented in this phase

- Quality Assurance engineers ensure that the development artifacts and content match expectations
  - Verification and Validation testers ensure that the capabilities are ready for delivery
  - Performance engineering specialists ensure that the software can perform
  - Software security specialists ensure that Information Assurance (IA) requirements are still met
- CM baseline delivery should bridge “traditional artifacts” with new artifacts
  - Some legacy customers may have never seen a model of the system
  - Mitigate this risk early on in the process. Communicate!
- Documentation delivery is vital to ending the vicious cycle of poor documentation

Information Assurance Implications

Deliver

Many systems have been deployed for decades

- Not hardened against IA threats
- "Grandfathered" acceptance due to age and necessity
- Assurance, certification and accreditation documentation may be incomplete or incorrect
  - STEPs, C&A plans, technical baselines, drawings

Be aware that there may be some serious IA implications with sustainment, and plan accordingly

- Customer may "expect" compliance, even though their RFP just references IA and DIACAP documents
- A lot of the newer IA guidance may have to be translated to apply to legacy systems and networks

Modernization may be the only solution to fixing these issues

- Hardware and software are enablers

Significant "Gotchas" in Sustainment

ASRC FEDERAL

Deliver

Under-estimation of effort

- Certain systems readily lend themselves to this sort of mistake
- Vigilance, rigor, and organization are keys to avoiding it
  - Remember to account for additional skillsets when legacy languages are concerned

Addition of updated or additional requirements

- Additional expensive and time-consuming requirements are sometimes levied on sustainers
- Many legacy systems have been "grand-fathered" over time

Availability of hardware, software and licenses

- eBay is running out of old motherboards!
- Software licenses for legacy products can be very expensive!
- Sustainment provides the best opportunity to modernize. Take advantage of technological enablers
  - Virtualization, modern emulators, porting capabilities

Questions?

Questions?