Welcome to the Webinar! Please note that we have moved to the ZOOM platform. Please join ZOOM audio (Voice over Internet) if you are able to connect. Otherwise, please see the webinar invitation for dial-in phone lines

INCOSE Webinar Series

Wednesday 15th November 2023 – Webinar 169

Systems engineering and software engineering: interactions among people, processes, and technologies

Richard (Dick) Fairley

Systems engineering and software engineering: interactions among people, processes, and technologies

contact me at

dickfairley@gmail.com

Some of my INCOSE involvements

- Joined INCOSE in 2012
- Currently Assistant Director for Accreditation of the INCOSE Academic Council
  - and a member of the ABET curriculum committee
- Member of the team that developed the ABET program criteria for accrediting systems engineering academic programs
  - and developed training material for systems engineering program evaluators in 6 ABET-EAC professional societies
- Member of the team that developed Version 1 of the Systems Engineering Body of Knowledge (SEBoK)
- Contributed two articles to the Systems Engineering Handbook V5
  - first articles on software engineering to appear in the SE Handbook

Other contributions

- Co-editor of the Software Engineering Body of Knowledge (SWEBOK, V3)
  - along with colleague Pierre Bourque
  - 28 contributing authors from several countries
- Team leader and primary author for development of the Software Engineering Competency Model (SWECOM)
- https://www.computer.org/volunteering/boards-and-committees/professional-educational-activities/software-engineering-competency-model

SWE, like SE, has a body of knowledge and a competency model

Today's Agenda

- Two key references
- SE & SWE relationships
- SE-SWE people interactions
- SE/SWE technology issues
- Hardware-software process issues

Two key references - 1

SEBoK (sebokwiki.org)

Part 3 System Lifecycle Models

Part 6 KA: Systems Engineering and Software Engineering

Five Topics in the Part 6 KA

1. Software Engineering in the Systems Engineering Life Cycle
   Tom Hilburn & Dick Fairley
2. The Nature of Software
   Dick Fairley
3. An Overview of the Guide to SWEBOK
   Dick Fairley & Pierre Bourque (V3 Editors)
4. Key Points a Systems Engineer Needs to Know about Software Engineering
   Dick Fairley and Alice Squires
5. Software Engineering Features - Models, Methods, Tools, Standards, and Metrics
   Tom Hilburn

Two key references - 2

My book

Fairley, R.E. 2019. *Systems Engineering of Software-Enabled Systems** Hoboken, New Jersey: John Wiley and Sons

*A software-enabled system is a system for which software enables a mission, business, or product

Systems engineering and software engineering

- Why do new systems and modified legacy systems increasingly rely on incorporated software?
  - because some system requirements and constraints can be more quickly and easily realized in software than in hardware
  - and because software is more malleable than is hardware
    - and is usually more easily modified that hardware as requirements, constraints, and hardware-software interfaces evolve
- SE and SWE are “intimately intertwined”*

*Boehm, B. W. "Integrating Software Engineering and Systems Engineering." *The Journal of INCOSE Vol. 1 (No. 1)*: pp. 147-151. 1994

SEBoK Part 6 KA: Systems Engineering and Software Engineering

Table 1. Adaptation of Methods Across SE and SWE (Fairley and Willshire 2011)
Reprinted with permission of Dick Fairley and Mary Jane Willshire.

Systems Engineering Methods Adapted to Software Engineering	Software Engineering Methods Adapted to Systems Engineering
• Stakeholder Analysis • Requirements Engineering • Functional Decomposition • Design Constraints • Architectural Design • Design Criteria • Design Tradeoffs • Interface Specification • Traceability • Configuration Management • Systematic Verification and Validation	• Model-Driven Development • UML-SysML • Use Cases • Object-Oriented Design • Iterative Development • Agile Methods • Continuous Integration • Process Modeling • Process Improvement • Incremental Verification and Validation

Software is malleable

- Composable and readable software (source code) is composed by typing symbols and clicking on icons
- A single missing, erroneous, or misplaced symbol, if undetected, can cause a large system to behave incorrectly or crash
  - for example, a “<” when a “>” was intended, or a missing ”;”
- Why might a *human error* be undetected?
  - because the syntax analyzer didn’t flag it
  - and because software testing is a sampling process
- An antidote: try to develop test cases that are representative of partitioned classes of functionality and behavior

Software engineers and software developers

- Most software engineers are current or past software developers
  - but not all software developers are software engineers*
- Software development requires concentrated attention to details
  - Some people are inherently detail oriented
  - and some are inherently “big-picture” thinkers
  - A competent software engineer has some of each ability

*Competent software developers are valuable engineering assets

Systems Engineers and Software Systems Engineers

Some differences

Systems engineers:

- engage in holistic systems thinking,
- pursue incremental system development, and
- rely on the expertise of other kinds of engineers

Software engineers:

- are more narrowly focused in their thinking,
- pursue iterative software development, and
- rely on the expertise of software developers

Some people-communication issues

Three fundamental issues that inhibit SEs and SWEs from effectively working together*

People-communication issues:

1. Different education and work experience backgrounds
2. Different incentives for success
3. Different usages of shared terminology

*Fairley, R.E. 2019. *Systems Engineering of Software-Enabled Systems,* Hoboken, New Jersey: John Wiley and Sons.

Differences in SE & SWE educations

- SEs typically have traditional engineering educations
  - based on continuous mathematics and quantified metrics
  - and “come up through the ranks” starting as traditional engineers
    - most don’t have software-awareness training or mentoring
- SWEs have a variety of educational backgrounds
  - typically based on discrete mathematics and computer science
  - and “come up through the ranks” starting as software coders and testers
    - most don’t have systems-awareness training or mentoring

Systems engineers' work experiences

- Hardware devices are procured as commodity items or fabricated as special purpose elements
  - Procurement delays can delay progress
  - special purpose elements are usually fabricated by technicians
    - sometimes by affiliated subcontractors
- Development of a system increment may require one or more months
- Development processes are sometimes dated and bureaucratic
  - sometimes caused by contractor-acquirer relations
- Holistic measures of success: on time, on budget, performance envelope scalability, adaptability, ease of integrating into a SoS, . . .
  - » are they prioritized?

Software developers' work experiences - 1

- Software can be implemented by software developers, reused from other systems and software libraries, and licensed from vendors
- Software is usually implemented by one or more small teams using weekly iteration cycles
- Newly developed software code can be stored in libraries for later reuse in other systems and contexts
  - competent software developers are aware of the code available for reuse and the contexts in which the code can be reused
    - and are always thinking (or should be thinking) "How can I make this software reusable in different contexts without violating the constraints of this application?"

Software developers' work experiences - 2

- In contrast to hardware, perfect copies of software can be repeatedly replicated with very little effort
  - but the perfect copies may be imperfect
- The primary incentive for success is usually software performance
  - response time, throughput, and use of computing resources
  - achieved at the risk of cutting corners that inhibit desirable features such as software security and future adaptability

desirable software features are tradeoffs with software performance

Communication failure antidotes

Antidotes for easing failures to communicate*:

- cross-training, mentorship, and relevant work experience
- lectures, workshops, short courses, and reading

Antidote deterrent:

I can't spare my valuable SE/SWE to learn these things

A recipe for disaster:

repeating the same processes and expecting different result

There is no sliver bullet*

*Frederick P. Brooks Jr, 1986. "No Silver Bullet – Essence and Accident in Software Engineering" *Proceedings of the IFIP Tenth International Conference tenth world Computing Conference.*

Use and misuse of terminology

- SEs and SWEs use (and misuse) the same terms with different meanings

Examples:

"incremental, iterative, design, capability, performance, review, prototype , verification and validation techniques , . . ."

Antidotes:

- organization-specific and project-specific Glossaries of Common Terms
- consistent use of terminology by respected opinion leaders and document writers

Antidote deterrent:

- I don't have the time or resources to involve my people to develop glossaries and train my people to use them

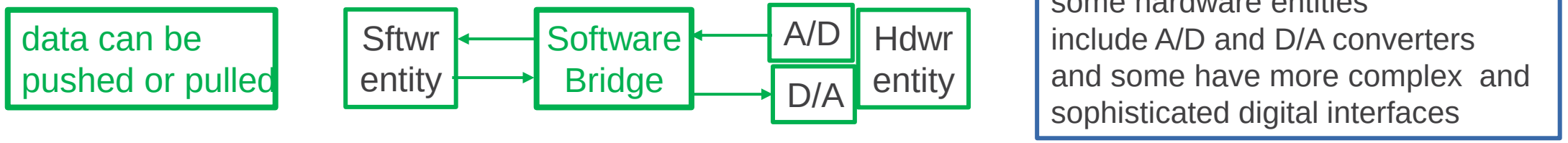
Two software technologies

1. hardware-software interface bridges
2. software design patterns

T1. Hardware-software interface bridges

- Hardware-software interfaces are the Achilles Heel of software-enabled system development
  - possible interface mismatches:
    - naming of interfaces and interface elements
    - numbers, types, and units of interface parameters
    - too many or too few parameters on one side of an interface
    - timing synchronization: race conditions
    - priorities of alarm signals and service interrupts

A simple hardware-software interface bridge

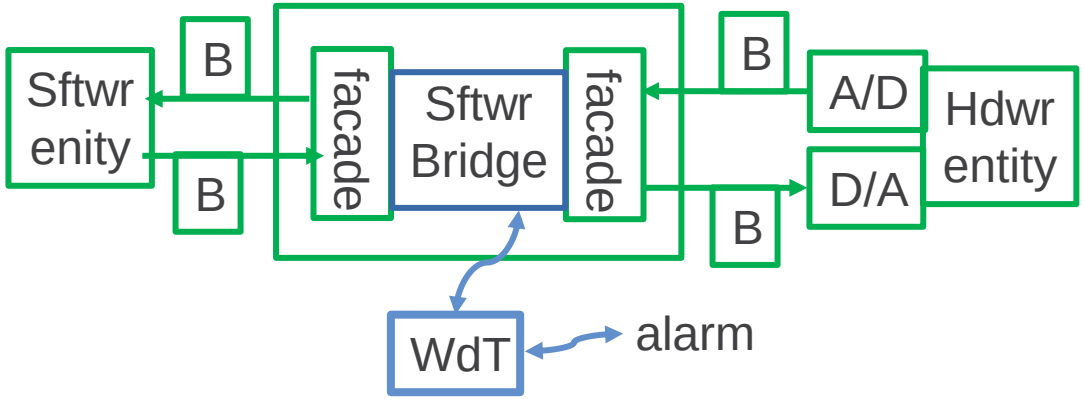


- A Software Bridge transforms software inputs into software outputs
- Sftwr entity: a system entity* copied from a software library or newly implemented for a particular use
- A/D and D/A: Analog to Digital and Digital to Analog converters
- Hdwr entity: a system entity that is not a software entity, a sentient being, or an element of the physical environment

*a *system entity* is any part of the system architecture

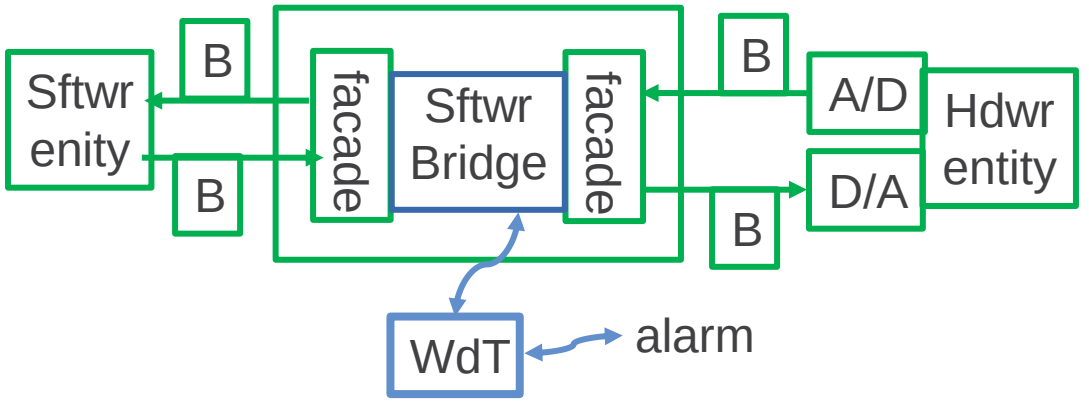
software bridges can provide hardware-software interfaces
without modifying the software entity or the hardware entity or their interfaces

A more complex hardware-software interface



- A *facade* has no executable code; it is a pass-through mask - facades can be used to mask some unwanted inputs and outputs without changing the Sftwr or Hdwr entity
  - to test or use some capabilities without allowing others to be activating
  - to tailor capabilities for different hardware or software entities
- a *buffer (B)* is an area of computer memory used as a temporary storage location
- a *watchdog timer (WdT)* can generate alarm signals when timing allocations are exceeded

Some observations:



1. A software bridge design pattern or tailorable bridge code may be copiable from a library
2. Tailorable facades, buffers, and timers are usually available from software code libraries
3. a bridge may be needed to connect the Sftwr Bridge to the Watchdog Timers (WdT)
4. WdTs are usually programmable with settable time durations

T2. Software design patterns

- A software library contains code to be used as is or as modified
- a design pattern is a best-practices template for solving a design problem within a given design context – and may be copiable from a library
- There is an annual software design patterns conference and a reference book
- The GoF design-patterns reference book includes 23 design patterns
  - seven of the patterns are the most-commonly used ones
- Competent software developers know (some) design patterns and when to apply them
- Design patterns also provide a common language for communication among software developers
  - e,g., “I’m using an MVC pattern for the display interface”
    - which may include a hardware-software bridge design pattern

Notes

1. Software bridges are not silver bullets
   - one side of a bridge may need an input parameter that is not provided by the output side of the bridge
   - the data from the output side of a bridge may require a lengthy computation that violates timing requirements
   - emergent behaviors may emerge
2. Design patterns are not silver bullets
   - A simpler solution may be adequate,
   - but it may not include features that enhance security or facilitate making enhancements and modification

But bridges and other design patterns are highly effective in many situations

Development processes

Differences in SE and SWE development methods

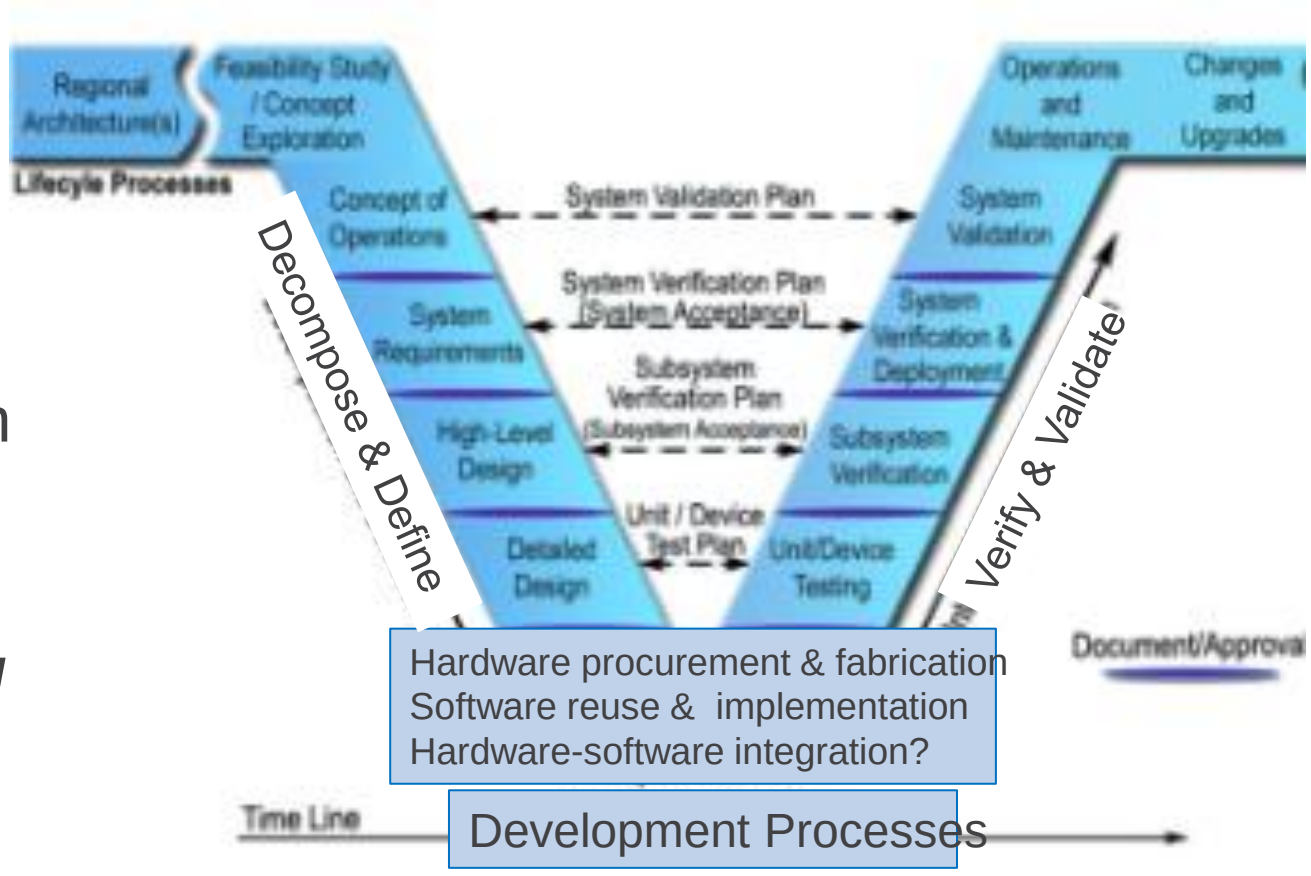
- Systems typically evolve in successive *increments*
  - development of a system increment may take a month or longer
  - and may involve multiple teams or multiple contractors
- Software typically evolves in successive *iterations*
  - iterations are typically completed weekly by one or more small teams
    - interpersonal communication among software developers limits individual team size not more than 5 software developers (perhaps 6 or 7)
    - schedule constraints may require multiple concurrent software development teams
      - with weekly integration, verification, and validation

Concurrent **hardware-software development**

Different processes for incremental hardware development and iterative software development do not sufficiently address integration of hardware and software

does a miracle happen?

or can software bridges and design patterns be used?



Synchronizing concurrent development processes?

- See chapters 5 to 9 of my book for a description of
  - I^3 The Integrated Iterative-Incremental Development Process
- The approach
  - always have a functioning something that can be demonstrated and that grows incrementally
    - a digital twin, a partial digital twin, a system skeleton, a backbone framework, a hardware subsystem or software being reused from a library or another system
    - some elements may be real, some may be prototypes,
    - some may be dummy interfaces, some may be simulations or emulations of system elements,
    - and some may be realized elements that replace digital twin elements

How to synchronize concurrent development processes?

1. software may be iteratively integrated into the evolving incremental system baseline to replace software prototypes and digital-twin elements
2. or hardware can be incrementally integrated into the evolving iterative software baseline to replace hardware prototypes and digital-twin elements
3. schedule frequent demonstrations of progress following V&V of a new baseline
   - *attended by authorized decision-making personnel*
   - with emphasis on the elements incrementally added or replaced
     - and on the interfaces among the new elements and existing elements
4. prepare reports of progress achieved and not achieved for each demonstration - *and don't hide the reports*
5. maintain a schedule of elements to be incrementally added and demonstrated, revised as necessary - *and don't hide the schedule of planned vs actual*

Does the I^3 development process work in practice?

- Yes, there are no unproven elements in the process
  - the contribution of I^3 is integration of proven techniques
- No, if the process is not followed as specified

disclaimer: the process has been shown to work on smaller-scale hardware-software projects but not on large systems programs

- Comments? Questions? please contact me at dickfairley@gmail.com