Systems engineering and software engineering: people, problem solving methods, technologies, and development processes
Part 2

prepared and presented by

Dick Fairley

dickfairley@gmail.com

A brief recap of Part 1

- Three key references
- Some fundamental issues
- Who are software engineers?
- SE and SWE problem solving methods

Three key references

I. SEBoK Part 3 System Lifecycle Models (sebokwiki.org)
 and SEBoK Part 6 Systems Engineering and Software Engineering

II. The Systems Engineering Handbook V5

III. My book

Systems Engineering of Software-Enabled Systems, Richard E. (Dick) Fairley, Wiley, 2019

Chapters 5-9 and Appx A&B

Software-enabled systems are systems for which software is essential in supporting missions, businesses, and products

SEBOK Part 6

I. Five Topics in the Part 6 KA: Systems Engineering and Software Engineering

1. Software Engineering in the Systems Engineering Life Cycle

   Tom Hilburn & Dick Fairley

2. The Nature of Software

   Alice Squires

2. An Overview of the SWEBOK Guide

   Dick Fairley & Pierre Bourque (V3 Editors); V4 being developed

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

Some fundamental issues

Seven fundamental issues that inhibit SEs and SWEs from effectively working together

1. Different education and work experience backgrounds
2. Different incentives for success
3. Different usages of shared terminology
4. Different ways of applying problem-solving techniques
5. Different development processes
6. Different approaches to developing hardware-software interfaces
7. The silo effect

Who are “software engineers?”

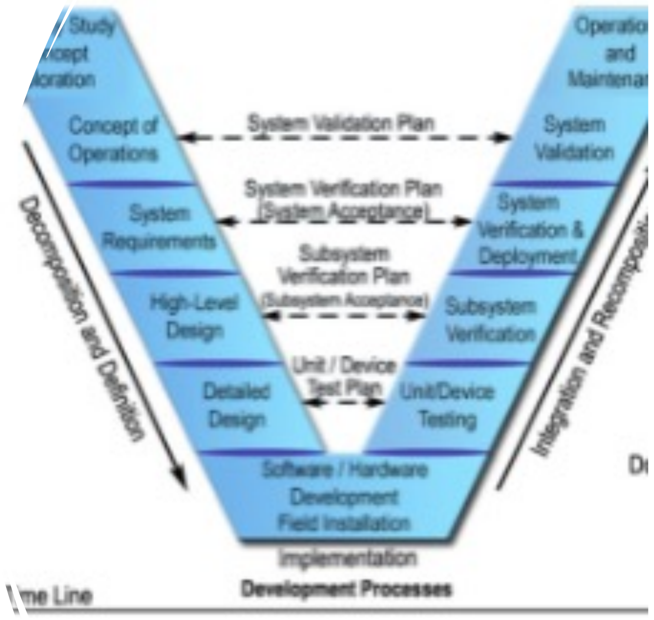
- The term “software engineer” is used with a variety of meanings
- See the software engineering competency model (SWECOM) https://www.computer.org/volunteering/boards-and-committees/professional-educational-activities/software-engineering-competency-model
- SWECOM includes 13 skill areas, skill categories, and activities at five levels of competency ranging from technician to senior software engineer*
  - can be used for (private?) self-assessment of strengths and weaknesses
    - and to council employees
  - to develop career paths and individual improvement plans
    - short course, academic courses, OJT, mentoring
  - can be used to assess project and organization capabilities and weaknesses

*SWECOM also includes the topics of requisite knowledge, cognitive skills, behavioral attributes, and related disciplines

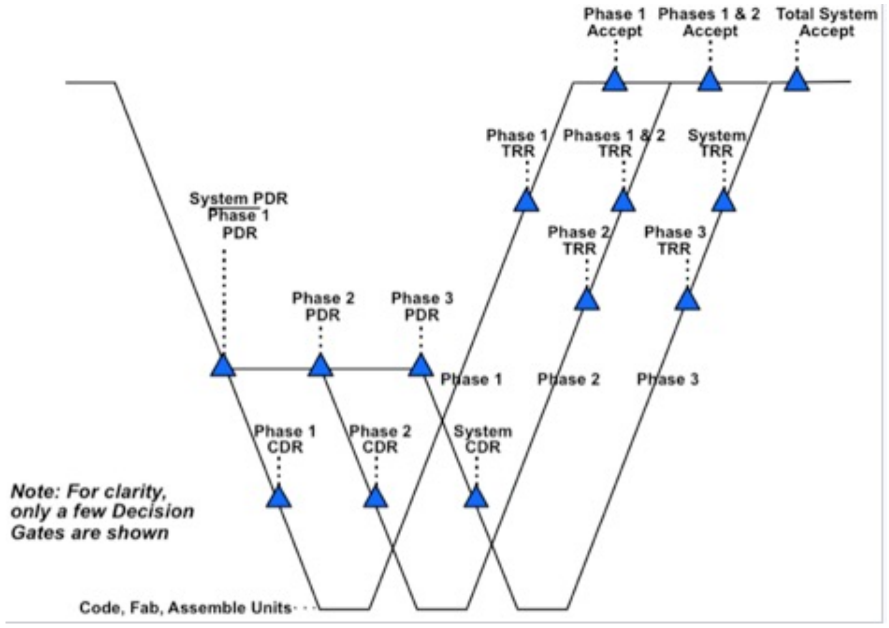
Systems engineering problem solving

- Systems engineers are holistic problem solvers
  - SEs focus is on the “big picture”
  - because many different constituents, technologies, technology experts, and rules and regulations must usually be accommodated

System developers often use Vee development models

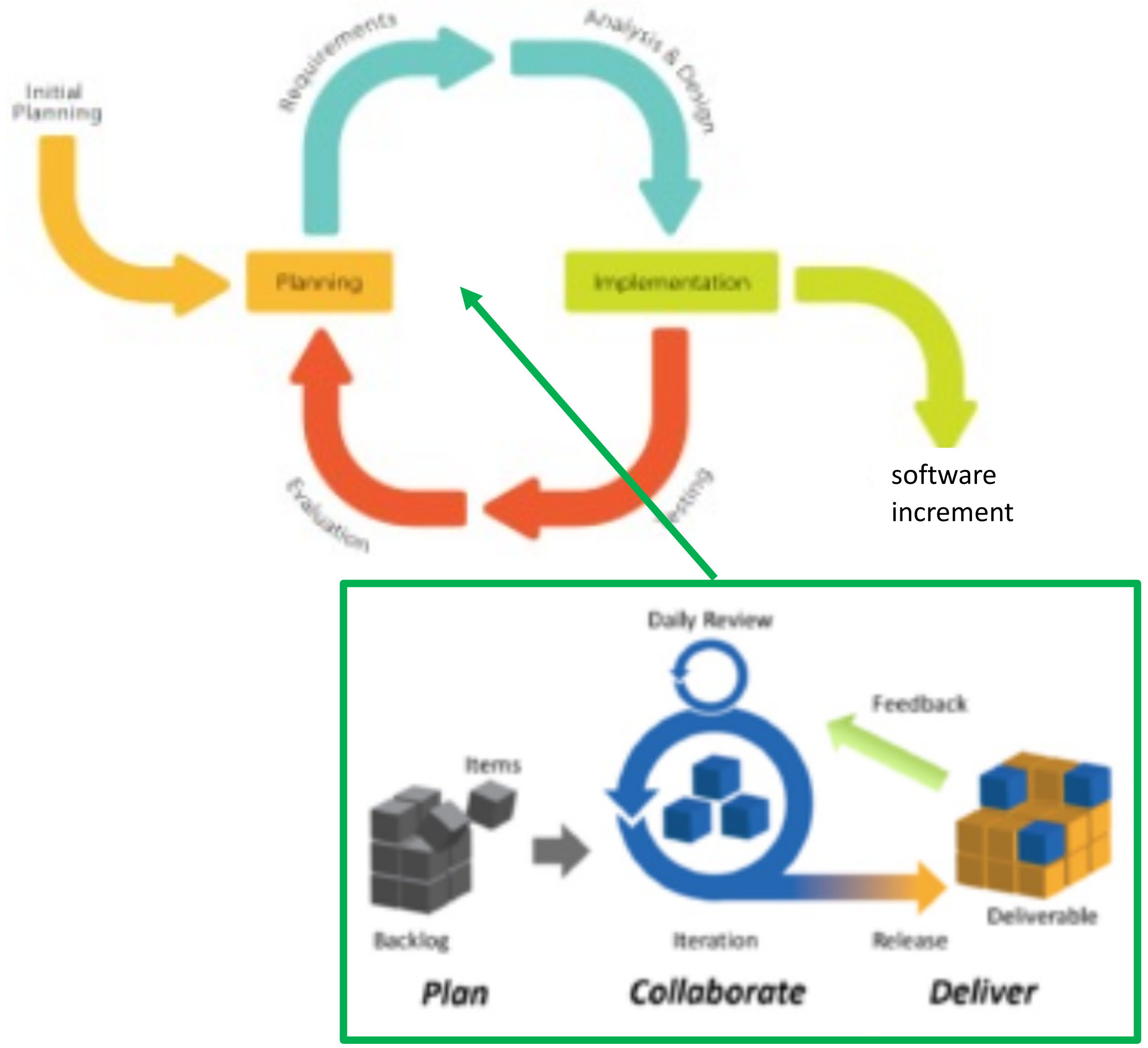


Hardware developers sometimes use incremental Vees that sometimes overlap



Software engineering iterative problem solving

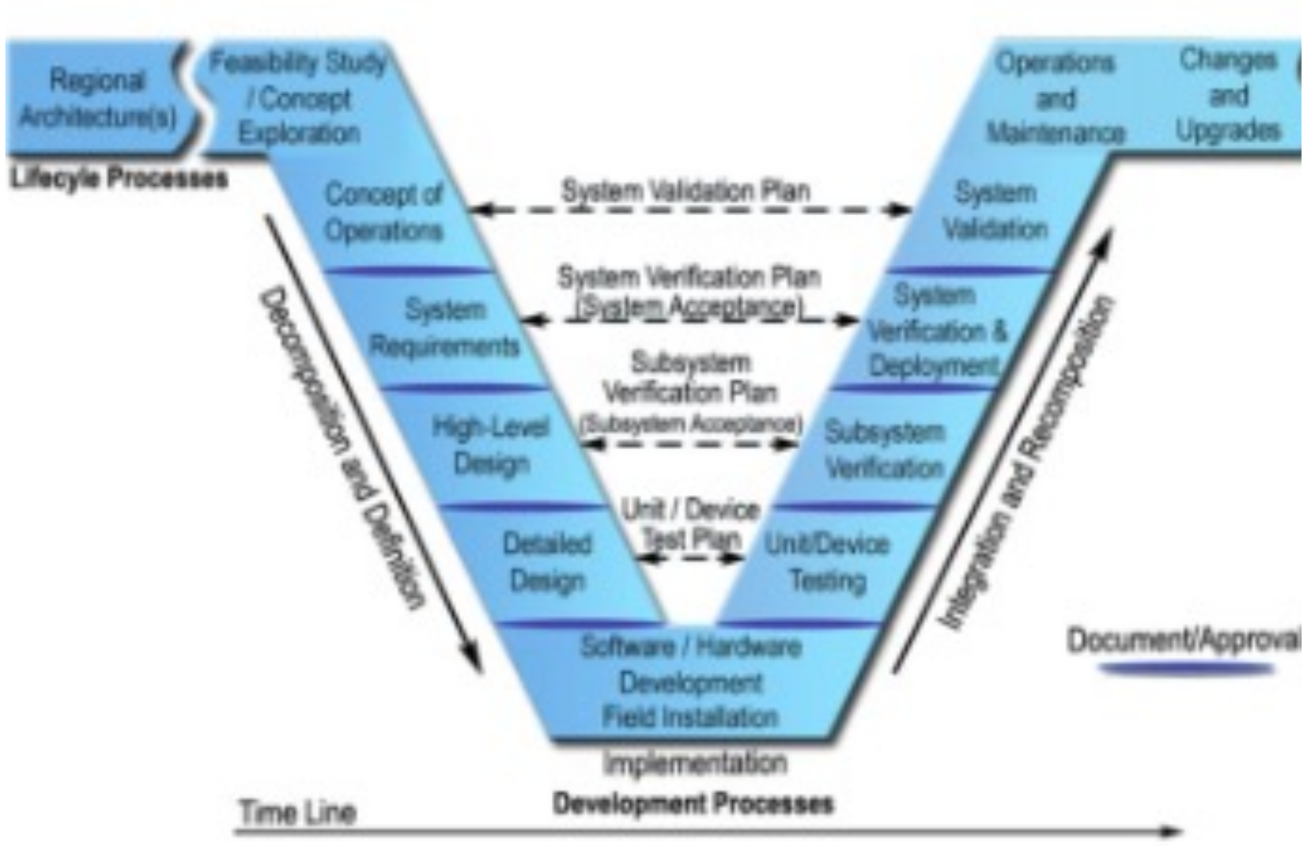
- Software engineers are detailed problem solvers
- Software increments are typically produced weekly and added to the evolving baseline of a system or subsystem
  - a 4-to-6-person team may do daily agile development a new baseline is then created after testing, correction, and demonstration*
- a 40-hour work week typically includes 4 hours planning, 32 hours of review, development and verification testing; and 4 hours (or more) of validation testing
- one person may do daily integration and testing against the baselined subsystem on a rotating basis



Dealing with hardware-software interfaces

NOTE: The Vee, Spiral, Scrum, and other approaches for incremental hardware development and iterative software development do not address incrementally integrating hardware and software during software-intensive systems development

- then a miracle happens?



"Implementation" is sometimes phrased as: code, fabricate, assemble

Hardware-software interfaces

- 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
    - human-user interface expectations
- Antidotes:
  - Shared Interface Control Documents (baselined and frequently updated)
  - And frequent demonstrations of incremental progress

How to synchronize concurrent development processes?

- How to synchronize concurrent incremental hardware and iterative software development processes
  - see chapters 5-9 of my book for a description of

    The Integrated Iterative-Incremental Development Model (I^3)
- An approach
  - always have a functioning something that can be demonstrated and that grows incrementally
    - a digital twin, a partial digital twin, a system skeleton or backbone, a hardware subsystem* or software being reused from another system
    - some elements may be real, some may be prototypes,
    - some may be dummy interfaces, some may be simulations of elements,
    - and some may be realized replacement elements for digital twins

*see https://zipcpu.com/blog/2020/01/13/reuse.html for Lessons in Hardware Reuse

Software reuse is easier because software is easier to modify (but not always easy)

Today’s Agenda

- SE-SWE communication inhibitors and antidotes
  - different educations
  - different work experiences
  - different usages of terminology
  - different success criteria

SE and SWE communication inhibitors*

Differences in educations

- SEs typically have traditional engineering educations
    - based on continuous mathematics and quantified metrics
    - and “come up through the ranks” starting as traditional engineers
        - some SEs have and some don’t have SE training and mentoring
- SWEs have a variety of educational backgrounds
    - typically based on discrete mathematics and computer science
        - or a masters degree conversion program
    - and “come up through the ranks” starting as programmers
        - most without SE awareness training or mentoring

Antidotes to ease failures to communicate:

- cross-training and mentorship
- readings, lectures, workshops, and short courses

* “what we have here is failure to communicate”
warden to prisoner Paul Newman in the movie *Cool Hand Luke*

Use and misuse of terminology

- SEs and SWEs use and misuse the same terms with different meanings
- Examples:
  “capability, performance, quality assurance, verification, validation, review, prototype , . . .”

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

SE and SWE communication inhibitors - 2

Hardware work experiences

- Hardware devices are fabricated or procured
  - as commodity items and special purpose (bespoke) entities
  - development increments may require one or several months
  - development processes are sometime dated and bureaucratic
    - sometimes based on acquirer-contractor relations
  - holistic measures for success: on time, on budget, performance envelope, scalability, adaptability, ease of integrating into a SoS . . .

SE and SWE communication inhibitors - 3

Software work experiences

- Software code is written by programmers and often stored in libraries
  - it is a malleable medium that is easy (too easy?) to change
  - in contrast to hardware, perfect copies can be replicated
  - development iterations often occur weekly
  - the incentive for success is often software performance
    - response time and use of resources
    - at the risk of cutting corners that inhibit security and future adaptability
- Why is the software always late?
  - ineffective development processes
  - late breaking changes to system requirements and design that are better accommodated by changing the software than changing the hardware

Questions? Comments?