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

prepared and presented by

Dick Fairley

dickfairley@gmail.com

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

"can't we all just get along?"
President Jack Nicholson in
the 1996 movie *Mars Attack*

Proposed Agenda

- Two key references
- Who are the “software engineers?”
  - The software engineering competency model
- SE and SWE problem solving methods
  - establishing hardware-software interfaces among system entities
  - incremental vs iterative development processes
- SE-SWE communication inhibitors
  - different educations
  - different work backgrounds
  - different usages of terminology
  - different success criteria

Two key references

I. SEBoK Part 3 System Lifecycle Models and Part 6 Systems Engineering and Software Engineering (sebokwiki.org)
Five Topics in the Part 6 Systems Engineering and Software Engineering KA:

1. Software Engineering in the Systems Engineering Life Cycle
   Tom Hilburn & Dick Fairley
1. The Nature of Software
2. An Overview of the SWEBOK Guide
   Dick Fairley & Pierre Bourque (V3 Editors); V4 being developed
1. Key Points a Systems Engineer Needs to Know about Software Engineering
   Dick Fairley and Alice Squires
2. Software Engineering Features - Models, Methods, Tools, Standards, and Metrics

II. My book

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

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

Who are the “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

Questions? Comments?

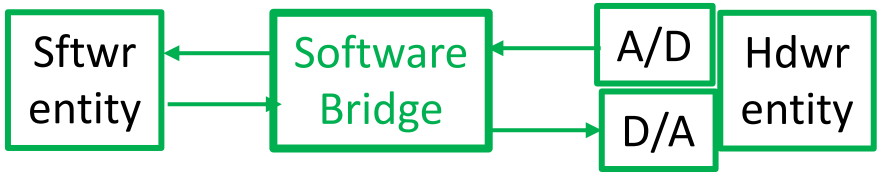
Hardware-software interfaces

- Hdwr-Sftwr 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 – difficult to reproduce; see the Therac-25 report* (1985-1987)
        - priorities of alarm signals and service interrupts
        - human-user interface expectations
- Antidotes:
    - Shared Interface Control Documents (baselined and frequently updated)
        - developed incrementally by participating responsible parties
        - maintained with appropriate levels of detail
        - with allocations of design responsibilities on both sides
    - And frequent demonstrations of incremental progress

*gsnag.com/blog/bug-day-race-condition-therac-25

A simple hardware-software interface

data can be pushed or pulled



some hardware entities include A/D and D/A converters and some have more complex digital interfaces

- Software Bridge transforms software inputs to software outputs; a *design pattern* can be used to design and construct a software bridge
- Sftwr entity: a system entity copied from a library or is newly designed for use *and for reuse*
- A/D and D/A: Analog to Digital and Digital to Analog converters
- Hdwr entity: a system entity that is not a software entity or a sentient being

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

Note: there are several kinds of software bridges; this bridge provides a hardware-software interface *without modifying either the software entity or the hardware entity*

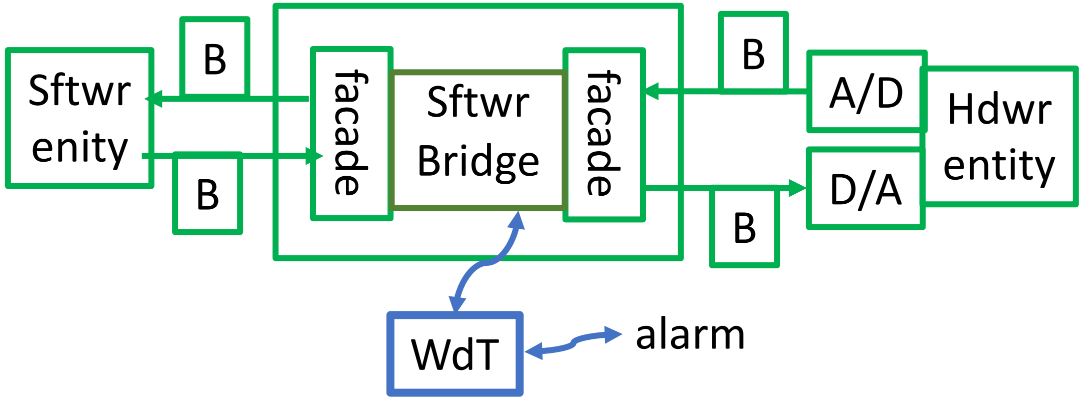
Software design patterns

1. software design patterns are templates for solving common software design problems within given design contexts
2. unlike a software library that contains code to be used right away, a design pattern is a best practices template for solving a problem
3. "best practice" usage is determined by widespread adoption; there is an annual design patterns conference and a reference book
4. The GoF design pattern reference book includes 23 design patterns which are classified in three categories: Creational, Structural and Behavioral patterns

   seven are regarded as the most widely used
5. competent software engineers know the design patterns and when to apply them
6. design patterns provide a common language for communication among software developers; e,g., "I'm using an MVC pattern for the user interface"

design patterns are not panaceas; a simpler solution may be adequate

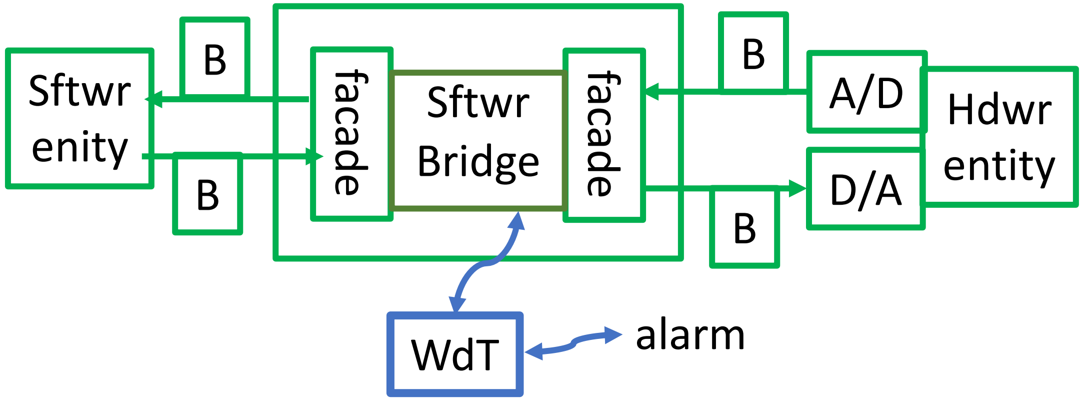
A more complex hardware-software interface



a hardware-software interface may include one or more bridges, facades, buffers, and timers

- a Sftwr *bridge* includes software to transform inputs & outputs as needed by a Sftwr or Hdwr entity; *and to correct interface mismatches on slide 17*
- 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 different clients
- 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:



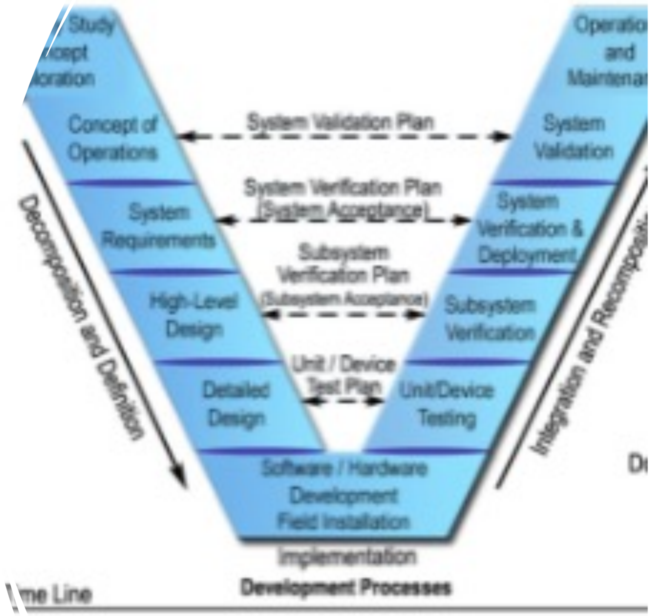
1. The hardware entity may have been procured off-the-shelf or bespoken
2. The software entity may be from a code library (perhaps modified) or newly written for a particular use *and perhaps to be stored in a library for later reuse*
3. A bridge design pattern or tailorable bridge code and may be copied from a library
4. Tailorable facades, buffers, and timers are usually available for copying from software code libraries; a bridge may be needed to connect the Sftwr Bridge to the WdT
5. Watchdog Timers can be programmed with elapsed time durations
6. Bridges can be used to bridge between all kinds of system elements: hardware-hardware, software-software, hardware-software, system elements-system environments, and internal HCI terminals to internal system elements and to external environmental elements
    - and can be modified for changing situations without changing other system elements

Questions? Comments?

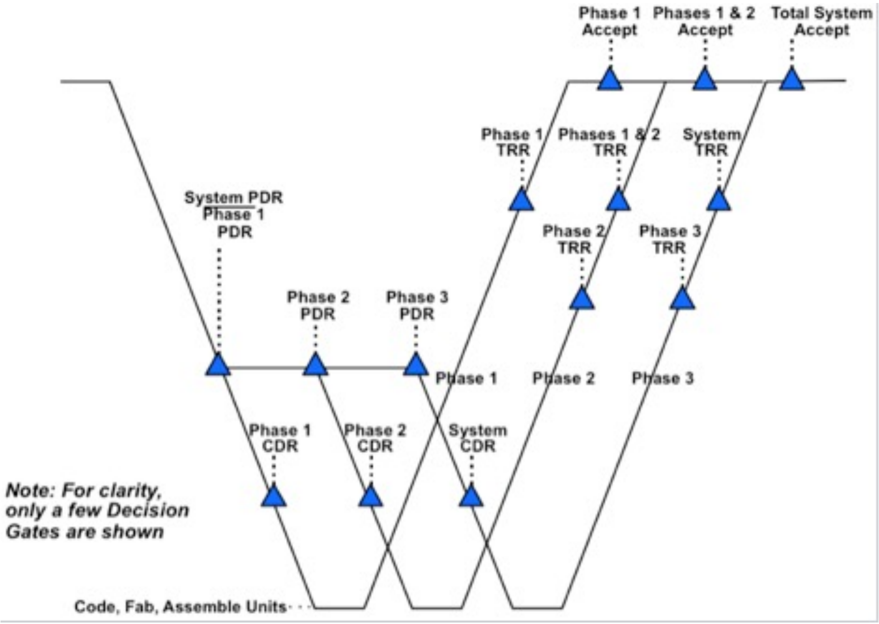
Systems engineering methods of problem solving

- 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



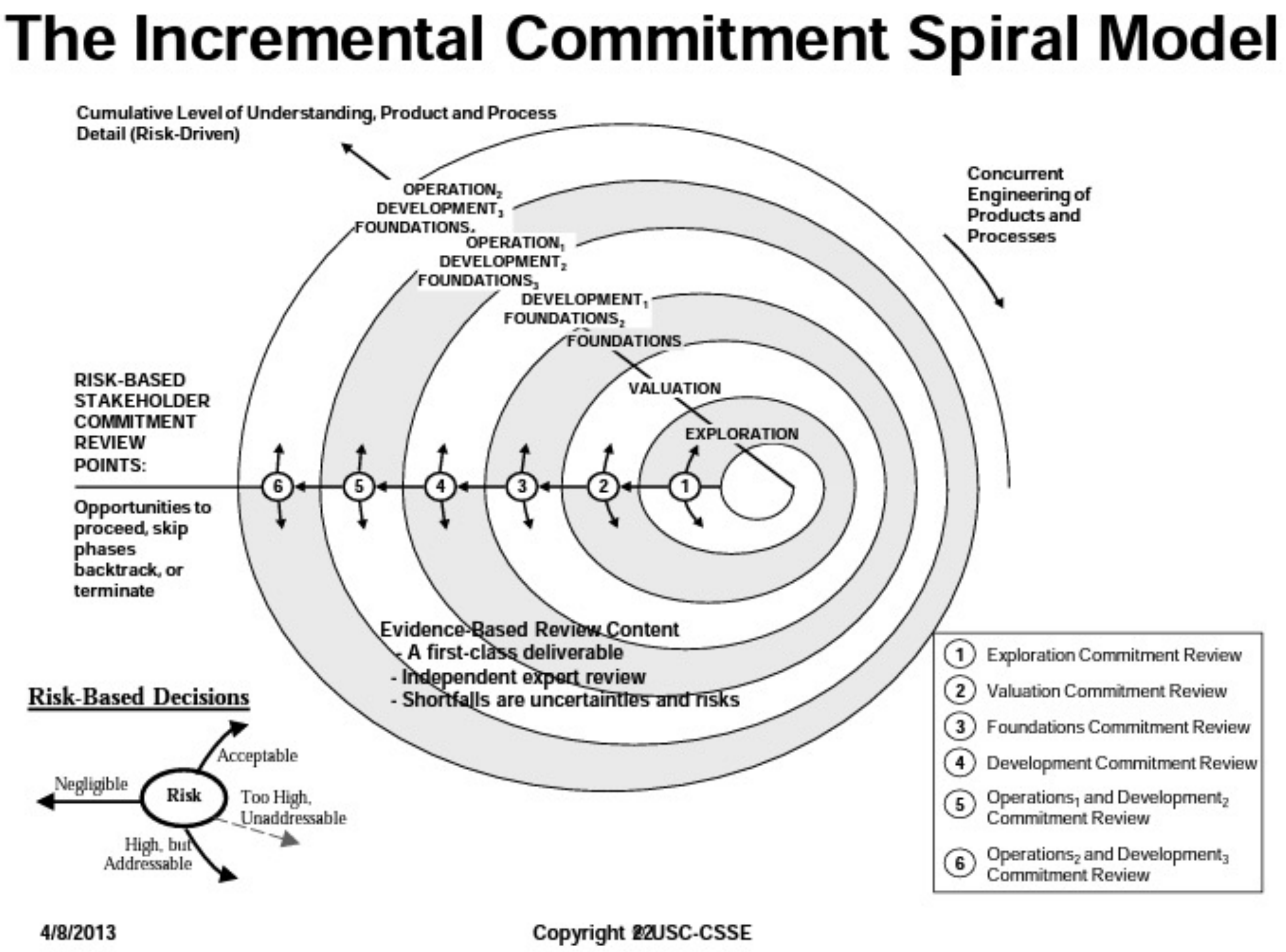
Systems engineering methods of problem solving - 2

- Systems engineers and software engineers sometimes use Spiral development processes
- The Incremental Commitment Spiral Model (ICSM) emphasizes evidence-based risk management and concurrent development

ICSM references:

Barry Boehm, ACM Webinar, 12/17/13 on the Internet

B. Boehm, *The Incremental Commitment Spiral Model*, Addison-Wesley Professional (June 3, 2014)



The Incremental Commitment Spiral Model

Evidence-Based Review Content
- A first-class deliverable
- Independent expert review
- Shortfalls are uncertainties and risks

1. Exploration Commitment Review
2. Valuation Commitment Review
3. Foundations Commitment Review
4. Development Commitment Review
5. Operations$_1$ and Development$_2$ Commitment Review
6. Operations$_2$ and Development$_3$ Commitment Review

4/8/2013

Copyright ©USC-CSSE 22

Software engineering methods of problem solving

- SWEs are detailed-oriented problem solvers
- SWEs focus on the pixels in the big picture

  (is there a big picture?)
- because extreme attention to details is required
  - a single omitted semicolon caused a large embedded program (and the system) to crash
  - three additional program statements were written correctly but included in the wrong order in a million-statement program
- why not detected during test and evaluation?
  - because software testing is a sampling process
    - the large number of combinatorial execution paths are data dependent
    - systems have been delivered with some untested software execution paths

NOTES

NOTE 1: sampling and destructive testing of hardware production runs and documented operational failures can be used to develop statistical reliability models

NOTE 2: software doesn't wear out (i.e., it doesn't age with repeated use)

  however, software can fail after repeated successful uses and after undocumented changes are made

    because testing is a sampling process and dynamics such as a rarely occurring race condition or a faulty execution path that has never been executed can cause system failure after successful use

1. software doesn't break; it's delivered broken
2. I can deliver it today if it doesn't have to work
3. it will be ready when it's ready*

  * some software developers are perfectionists

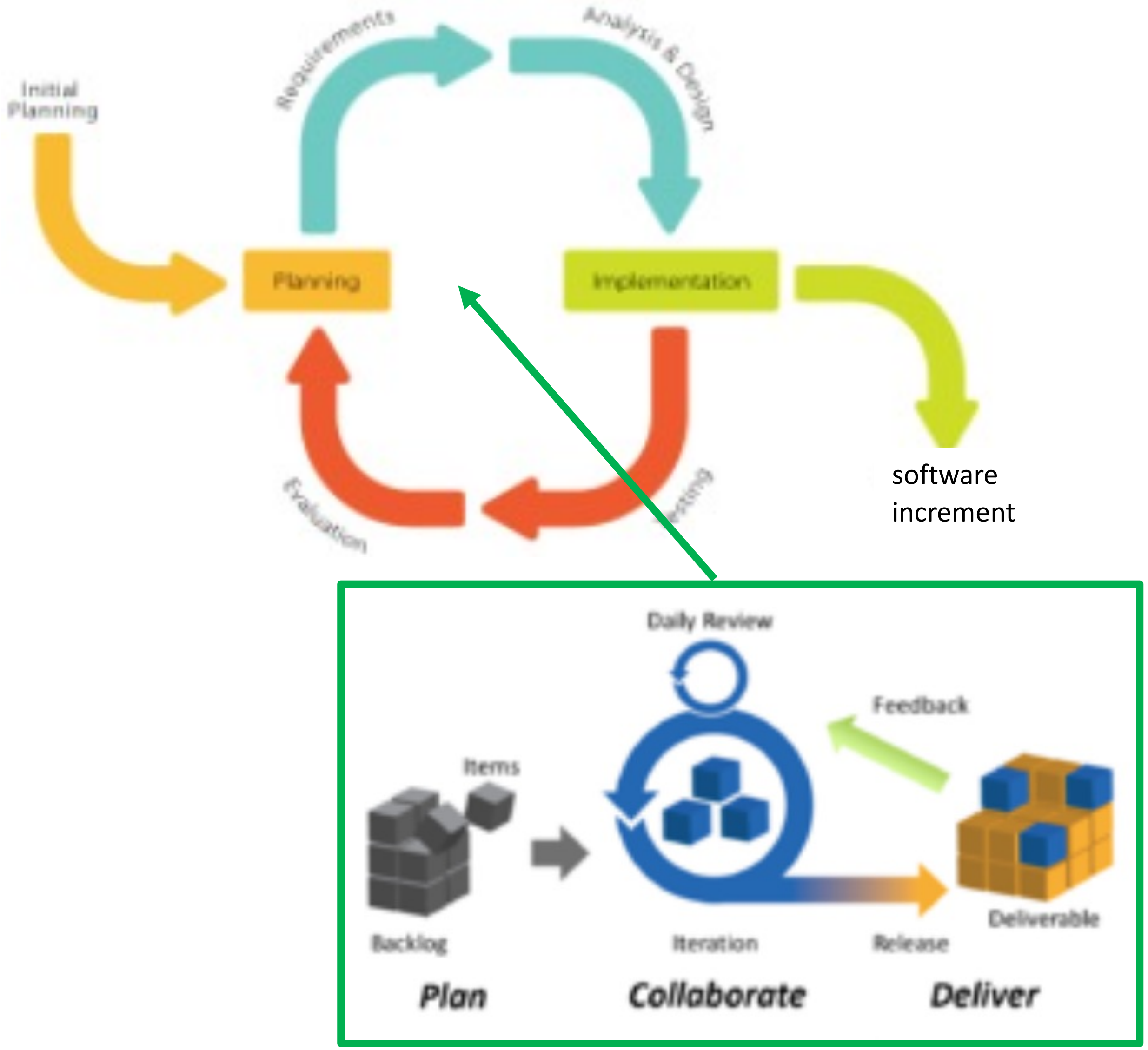
How to envision a million lines of software source code?

- A ream of 8½ x 11 20 lb. paper has 500 sheets per ream and is 2 inches thick
- If printed at 50 lines per page, a million-line program would be a stack of paper 6.67 feet high
- and somewhere in that stack of paper is a semicolon was accidentally omitted
  - and the syntax analyzer and testing didn’t catch it
- or three correct statements were added in the wrong order
  - the syntax was OK; but the pragmatics of the three statements taken together were incorrect
    - *and insufficient certification testing didn’t catch it; it was only 3 lines of code, each written correctly*

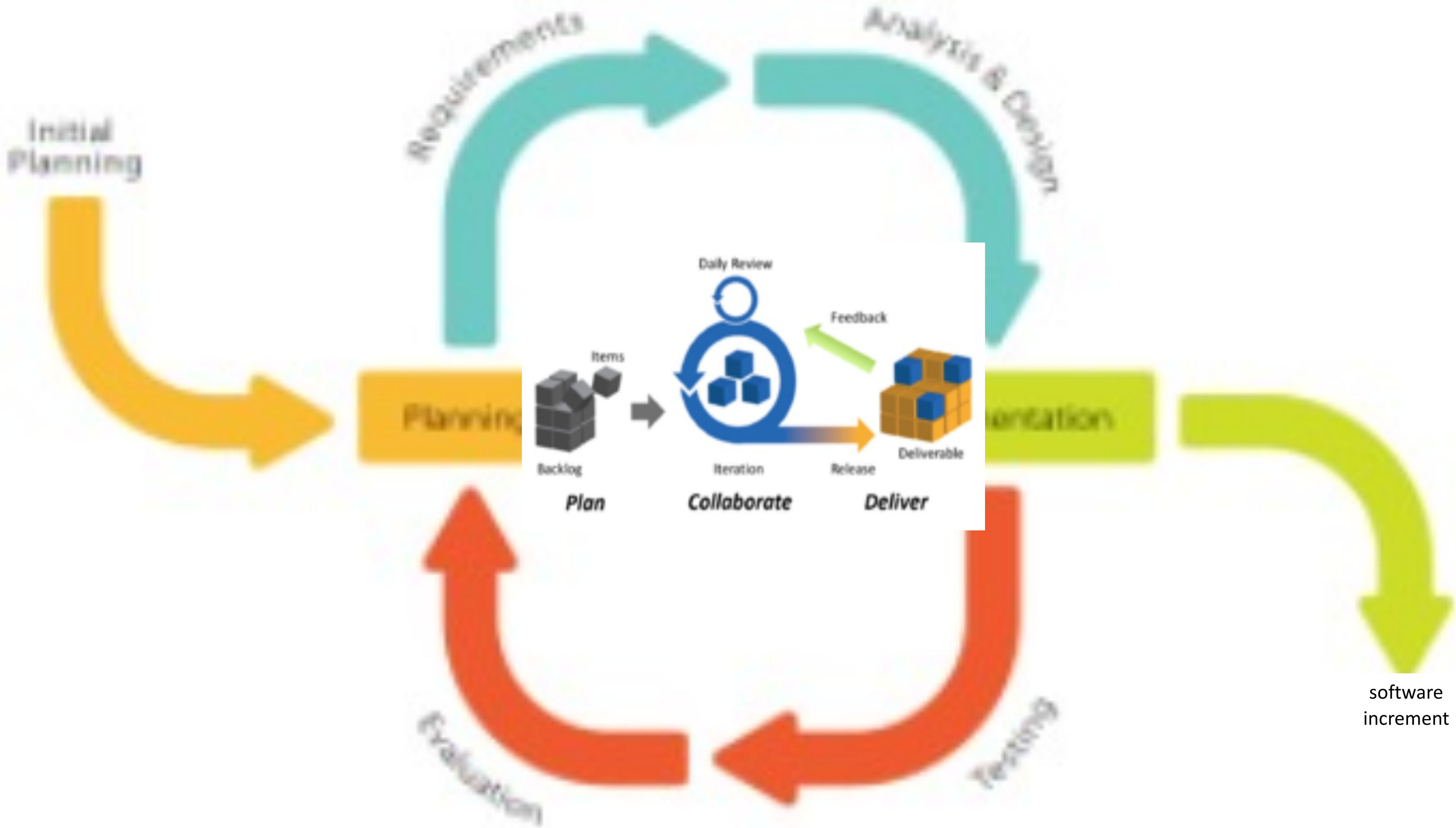
Software developers often use iterative development models

Software developers often use iterative development models

- 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



software increment

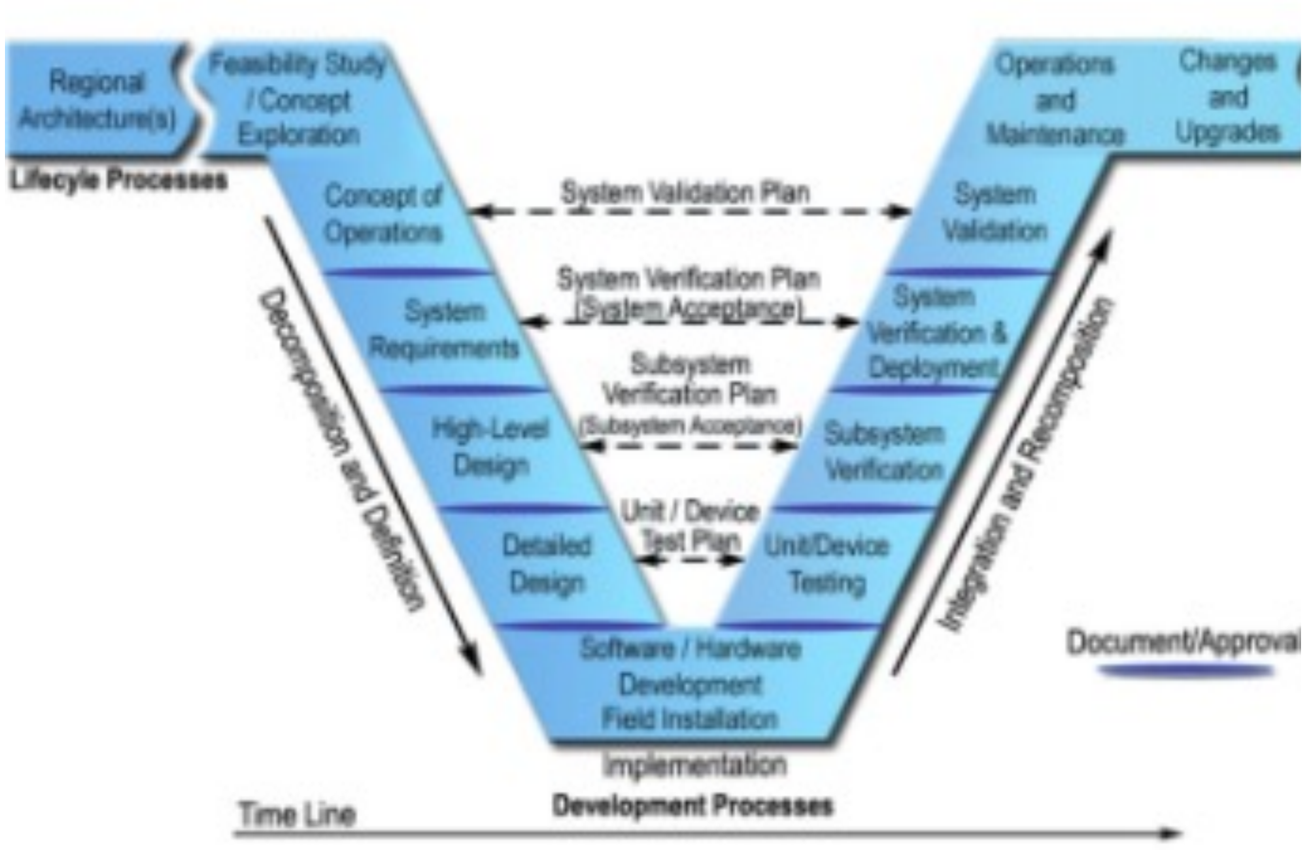


software
increment

Concurrent methods of problem solving

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

- then a miracle happens?



“Implement” is sometimes phrased as: code, fabricate, assemble

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)

How to synchronize concurrent processes? - 2

1. software developed iteratively (probably on shorter cycles than incremental hardware development) can be integrated into the evolving Vee incremental system baseline
2. or hardware (with appropriate interfaces) can be integrated into the evolving iterative software baseline to replace hardware prototypes and digital-twin elements
3. schedule frequent demonstrations of progress
   - attended by appropriate decision-making personnel
   - with emphasis on the elements incrementally added or replaced
   - and 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*

Questions? Comments?

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

Software work experiences

- Software code is written by programmers and sometimes 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

SE and SWE communication inhibitors - 3

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 . . .

Questions? Comments?