A Framework for Integrating Systems and Software Engineering

NDIA Systems Engineering Conference
San Diego, California

Art Pyster
art.pyster@stevens.edu
Richard Turner
richard.turner@stevens.edu

October 21, 2008

Agenda

- Rationale: Why integrate systems and software engineering?
- Touchpoint: A framework
- Initial Results
- Next steps

Rationale: Assertions

- ***Interdependent*** systems are those where:
  - A "major" portion of the capabilities/value of the system is delivered through software
  - A "major" portion of system quality attributes "largely" depend on software (safety, security, agility, reliability, availability, resilience,...)
- Today most high value systems are interdependent; that percentage is increasing
- In these systems, nearly all important decisions require equal consideration of software engineering and systems engineering expertise
  - Technical, management, personnel and customer concerns are included
- But, what does it mean to integrate SE and SwE?

Rationale: Questions needing answers

1. What outcomes do we expect from SE/SwE integration?
   - Does integration reduce key risks?
2. How do you measure integration or it's outcomes?
3. ***How and why do the SwE and SE activities conflict, complicate, or reinforce each other?***
4. How much integration is needed?
   - What is the scope of integration (development, operations, business areas…)?
   - Is more integration always better?
   - Is integration domain- or application-dependent?
5. Why haven't IPTs or CMMI solved this problem?

Rationale: Barriers to integration

- Historical context and vestigial prejudices
  - SE and SwE cultures are significantly different
  - SE and SwE have different educational backgrounds
  - SE and SwE vocabularies are similar but meanings differ
- SE and SwE process implementations are often incompatible (e.g. V versus spiral)
- SE and SwE may use the same tools differently (UML)
- No language to discuss integration of SE and SwE

Rationale: Issues needing to be addressed

1. ***Vocabulary*.** There is no precise way to talk about the integration of systems and software engineering.
2. ***Measurement*.** There is no precise way to talk about *how much* integration there is between systems and software engineering in a particular situation.
3. ***Entanglement*.** The complexity of the disciplines makes it difficult to identify where software and systems engineering touch.
4. **Value.** There is no comprehensive list of benefits that can be achieved by integrating systems and software engineering nor is there an understanding of the associated costs.

Touchpoint

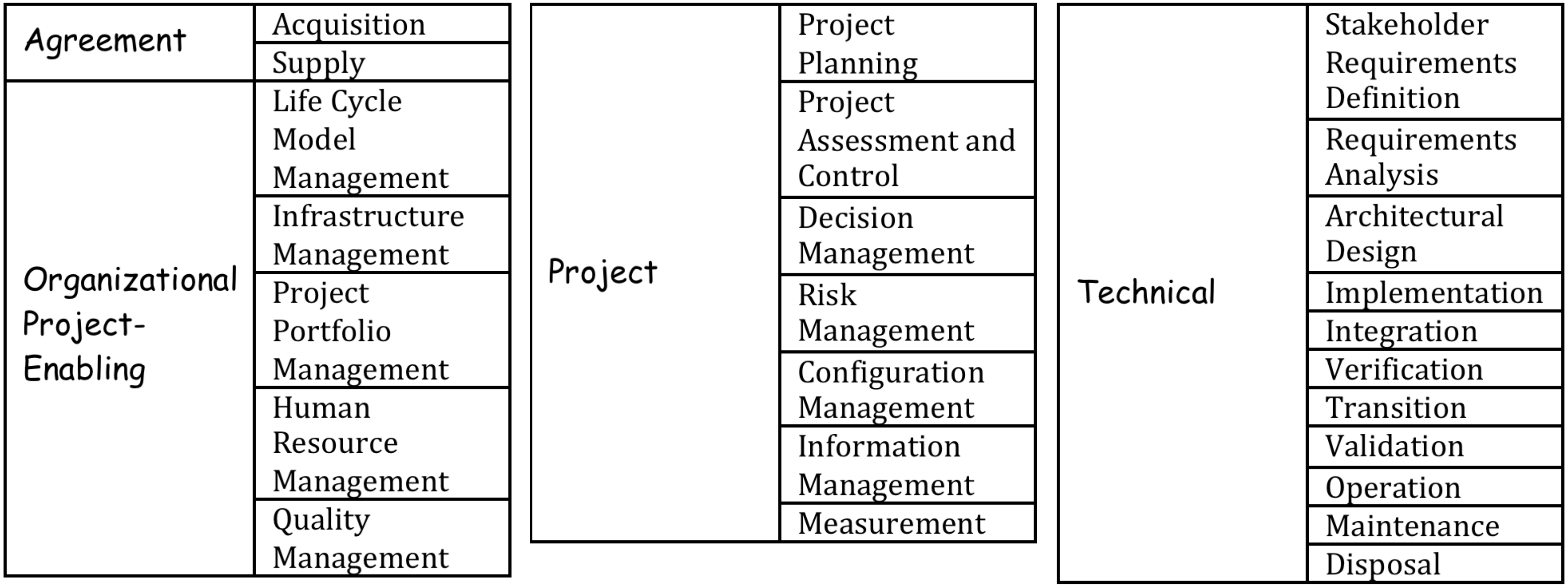
- A framework to support the discussion of SE/SwE integration
- Simple and (seemingly) robust
- Provides a way to describe integration at the practitioner level
- Describes touchpoints where the two disciplines interact
- May help to describe the degree of "integratedness"

Touchpoint Framework: Components

- ***Processes.*** The ordered activities that define the systems and software engineering disciplines
- ***Touchpoints (TPs).*** The two discipline’s processes *touch* when interactions between their constituent activities affect program risk or value – positively or negatively.
- ***Faults.*** A touchpoint may exist, but the process or activity may fail to produce its maximum value.
- ***Resolution Strategies (RSs).*** For each fault, there may be one or more actions that will eliminate the fault or reduce its impact.

Touchpoint Framework: Processes

- *ISO 15288* provides “harmonized” systems and software engineering processes
- Agreement, Organizational Project-enabling, Project, and Technical processes



Agreement	Acquisition
	Supply
Organizational Project-Enabling	Life Cycle Model Management
	Infrastructure Management
	Project Portfolio Management
	Human Resource Management
	Quality Management

Project	Project Planning
	Project Assessment and Control
	Decision Management
	Risk Management
	Configuration Management
	Information Management
	Measurement

Technical	Stakeholder Requirements Definition
	Requirements Analysis
	Architectural Design
	Implementation
	Integration
	Verification
	Transition
	Validation
	Operation
	Maintenance
	Disposal

Touchpoint Framework: Faults

- Gap
  - Logically, there should be an interaction between the corresponding SE and SwE processes, but the processes do not include one. A needed activity is therefore performed poorly, or not performed at all.
- Clash
  - One or more activities in each of the two corresponding SE and SwE processes produce are incompatible and result in inconsistent results or inconsistent actions.
- Waste
  - Activities in the two corresponding SE and SwE processes independently expend resources that produce the same result or take the same action with no added benefit to the program

Touchpoint Framework: Faults - Clashes

- Vocabulary
  - SE/SW activities use the same terminology with different meanings, or terms not recognized by the other, making communication harder
    - Example: Object-oriented terminology
- Value
  - Software and systems engineers in an organization or program value different process characteristics
    - Example: Stability of baselines
- Mental Model
  - Software and systems engineers think differently about how to carry out process activities
    - Example: “part-of” relationships vs. “uses” relationships.

Touchpoint Framework: Example TP

Process	Touchpoint	Fault	Type
Architectural Design	Systems architectures include significant software components to deliver critical capability	*Software-engineering architectures define layers of related functionality, while most systems-engineering methods are hierarchical structures.*	Clash – Mental Model

Example from pilot research

Touchpoint Framework: Resolution Strategies

- There is a desire to fix faults, especially those with high impact on risk or value.
- For each fault, there may be one or more resolution strategies, which, when executed well, will eliminate the fault or at least reduce its impact.
  - In some cases, resolution strategies are known and just need to be applied
  - On the other hand, resolving some faults will require research
- Resolution strategies are grouped into four traditional categories: *process, people, environment,* and *technology*. Any number of resolution strategies in each category is possible for a fault.

Touchpoint Framework: Example RSs

Process	Touchpoint	Fault	Type
Architectural Design	Systems architectures include significant software components to deliver critical capability	*Software-engineering architectures define layers of related functionality, while most systems-engineering methods are hierarchical structures.*	Clash – Mental Model

Resolution Strategy	Category
Research must be conducted to resolve the clash between object-oriented and structured methods. Maier provides some of the best research in this area.	Technology
Design software architecture to look just like system architecture. Make it easy for a system architect to understand. (SW systems mirror HW systems, e.g. relays, motors, etc). Then SW helps the system architect understand things in better detail.	*Process*
Middleware may be able to bridge the gap.	*Technology*

Examples from pilot research

Touchpoint Framework: Measurement

- Provides a way to measure *how much* integration has been achieved and *how good* that integration is.
- The amount of integration is simply the total number of touchpoints in the implementation of the 25 processes – a higher number indicates more integration.
  - A somewhat more sophisticated approach associates a weight with each touchpoint to reflect its potential impact on program risk or value.
- The number of faults determines integration quality.
  - Faults can also be weighted based on their consequence.
- A fault that severely impacts an important touchpoint would be of far greater consequence than a fault that barely impacts a minor touchpoint.

Initial research: Piloting

- Process activities at the "touchpoint" level are generally not found in available traditional documentation (standard processes, WBS, plans)
  - Often technical management/practitioner activities
- Approach – interview SE and SwE leadership
  - Identified ~10 programs through OSD AT&L and NDIA
  - Interviewed each program to identify touchpoints, faults, resolution strategies and challenges; rigid "no attribution" policy
- Compared interview findings with the systemic analysis findings of AT&L/SSE Program Support Assessments

Piloting Results

- Touchpoint elements (TPs, Faults, RSs) identified by Systemic Analysis Category

Category	Elements	No. of Projects
Architecture	12	6
CM	1	1
EVM	2	2
Human Capital	4	2
Process Planning	3	3
Requirements	23	10
Risk Management	2	2
System Integration	4	4
Software Metrics (Visibility)	4	3

Piloting Results

- Touchpoint elements not in Systemic Analysis Category

Category	Elements	No. of Projects
Contracting	4	3
Life Cycle	7	4
Technical Reviews	2	2

Sample Architectural Design Process Findings

Touchpoint	Fault	Type
Architecture concept	Underutilized software capability	Gap

Resolution Strategy	Category
Concept development should be performed jointly and careful trades made that reflect HW and SW capabilities, strengths, and weaknesses	Process

Touchpoint	Fault	Type
Meeting non-functional requirements	HW reliability numbers are calculated to many decimal places, and include the contributions of very low-level WBS components. SW reliability is not understood and so ignored.	Gap

Resolution Strategy	Category
Research in integrated reliability approaches is needed	Technology
Train systems and reliability engineers to understand software reliability	People

From pilot research
Authors' suggestion

Sample Requirements Analysis Process Findings

Touchpoint	Fault	Type
Software Requirements	SW specifications that limit trade space	Clash – Mental Model

Resolution Strategy	Category
Define software requirements in terms of "what" not "how."	Process
SE and SW collaborate in the development of software requirements	Process

Touchpoint	Fault	Type
Requirement Maturation	The difference in speed of maturation between HW and SW requirements causes tension between SEs and SwEs.	Clash – Mental Model

Resolution Strategy	Category
Requirements management tools and processes need to better support iterative approaches to requirements maturation.	Technology

From pilot research

Authors' suggestion

Sample Life Cycle Management Process Finding

Touchpoint	Fault	Type
SE and SW life cycles	Life cycle speeds differ causing perceived architecture instability and schedule coordination problems	Clash – Value

Resolution Strategy	Category
Involve SEs in software projects using iterative life cycles to gain comfort and trust.	People

From pilot research
Authors' suggestion

Conclusions and Next steps

- Framework seems useful
- Need much more data
  - More programs
  - More variety
- Refine and extend initial findings with new data
- Create products that make findings useful to programs

Questions and Discussion