WELCOME!

INCOSE Enchantment Chapter Monthly Meeting

INCOSE

ENCHANTMENT CHAPTER

We’re glad you’re here.

We respectfully request:

- Mute your audio when you are not speaking
- *6 toggle or in GlobalMeet left-side, your name

Discussion and questions are encouraged!

Put questions in the chat box or unmute yourself to speak up.

Meeting Materials

Slide presentations can be downloaded prior to start of the meeting from the Meeting Materials page of our website:

https://www.incose.org/incose-member-resources/chapters-groups/ChapterSites/enchantment/resources/meeting-materials

If recording is authorized by speaker, the video will be posted at the link above within 24 hours.

SEP Training

CSEP Courses by *Certification Training International:*

CTI currently is offering online course offerings, see

https://certificationtraining-int.com/incose-sep-exam-prep-course/

Our chapter has two SEP mentors:

Ann Hodges alhodge@sandia.gov

Heidi Hahn drsquirt@outlook.com

Upcoming meetings

- July 14, 2021: Dr. Dave Peercy, Education as a System of Systems
- August 11, 2021: Pat Foley, WBS Integration with an Effective Schedule
- September 8, 2021: Brian Kennedy, Leveraging Set-Based Practices to Enable Efficient Concurrency in Large Systems and Systems-of-Systems Engineering

Introductions

- Please type your name, position, and organization in the Chat window

Photo by Adam Solomon on Unsplash

Survey

The link for the online survey for this meeting is

- www.surveymonkey.com/r/2021_06_MeetingEval

Your feedback is important!

Enchantment Chapter Monthly Meeting

Interface Management, The Neglected Orphan of Systems Engineering

Abstract: Every Interface is an opportunity to lose information, time, control and / or money through contention between stakeholders at either end. There are many issues surrounding Interface management, which are relatively unexplored in the engineering literature. Interface management is perceived as a critical skill in the engineering of successful systems (INCOSE TP-2018-002-01.0), but finding useful material on the subject proves elusive. It is not that there is a gap in the collective Body of Knowledge (BoK) – but there is definitely a gap in the *documented* BoK. This paper explores some of the characteristics of this gap, and outlines some of the key concepts in best practice. Along the way, the differences between best practice for interfaces and best perceived practice for architecting systems are noted, and recommendations for changes in approach are given.

Download recording from the Library at www.incose.org/enchantment

NOTE: This meeting will be recorded

Speaker Bio

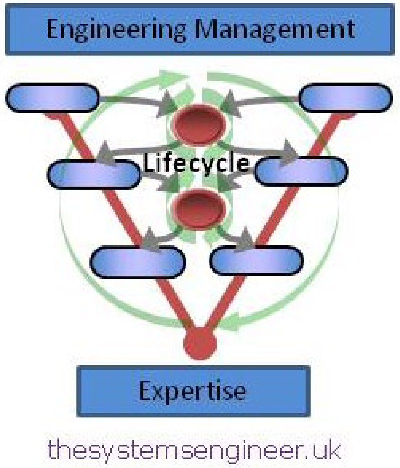
INCOSE ENCHANTMENT CHAPTER

Paul Davies supposedly retired in early 2014, but soon realized he needed to give something back to the systems engineering community and help mentor the next generation of practitioners. An experienced systems engineer with a sound track record in delivering successful projects over thirty years in the defense and aerospace industry, six years in the nuclear industry, and a couple of years in rail, he has a wealth of diverse experience to call on. Paul has conducted training courses and workshops in requirements, interface management, verification and validation, systems engineering management, competence assessment, and SE return on investment, with very positive feedback.

Interface Management – The Neglected Orphan of Systems Engineering

Paul Davies

paul@thesystemsengineer.uk

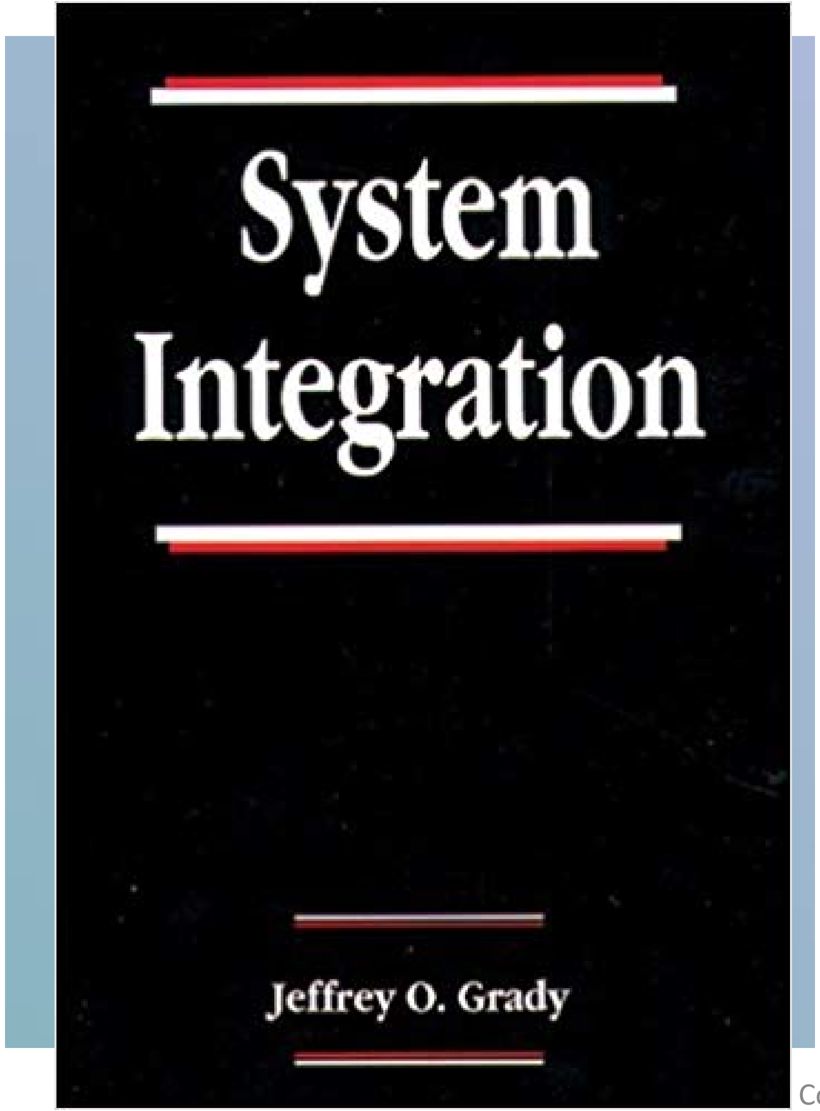


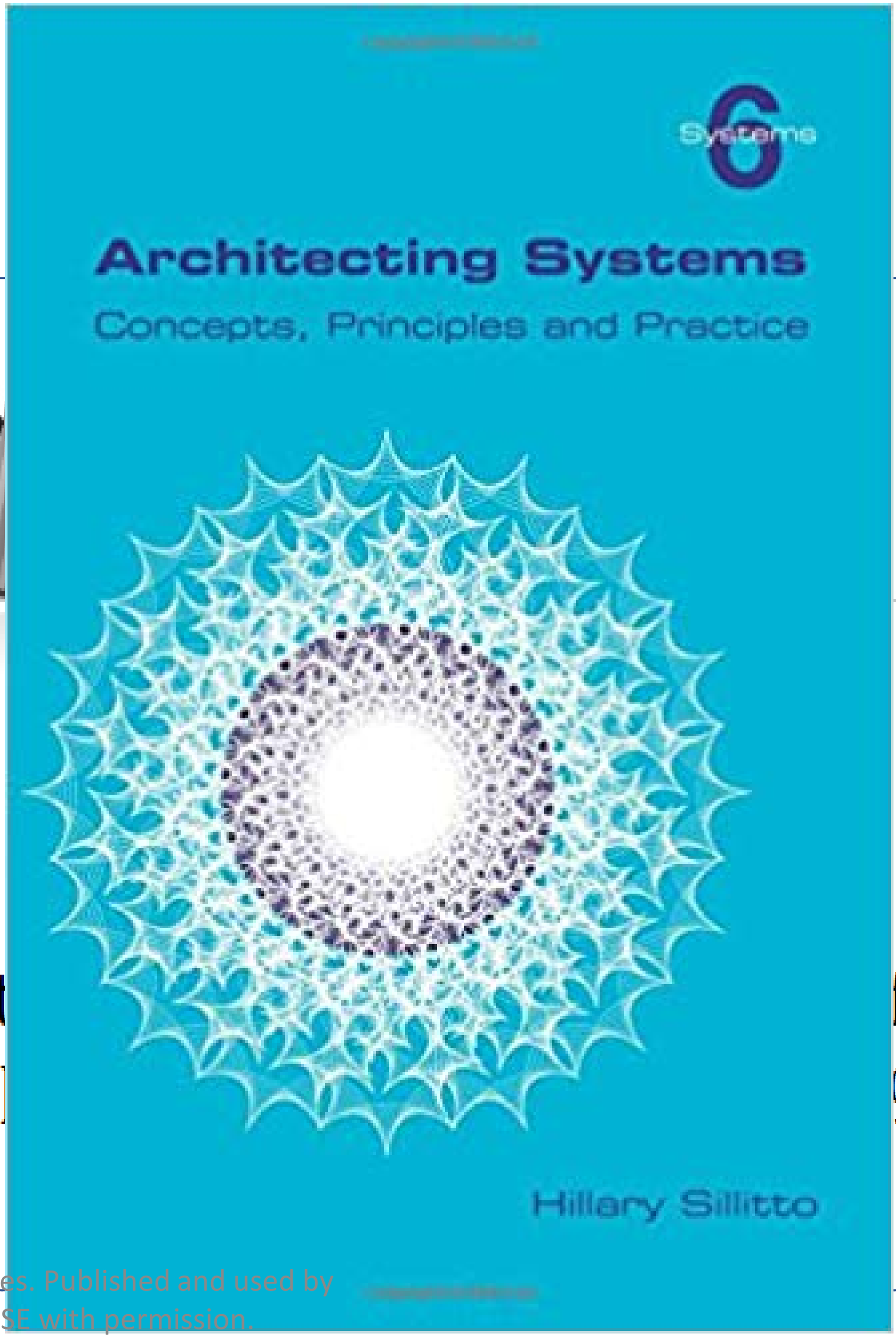
Aims and Outline

- To challenge the perception of an engineer as someone who ignores the world outside their System Element
- To remove the excuse "There's no training or guidance on interfaces"
- To left-shift the consideration of interfaces in architecting

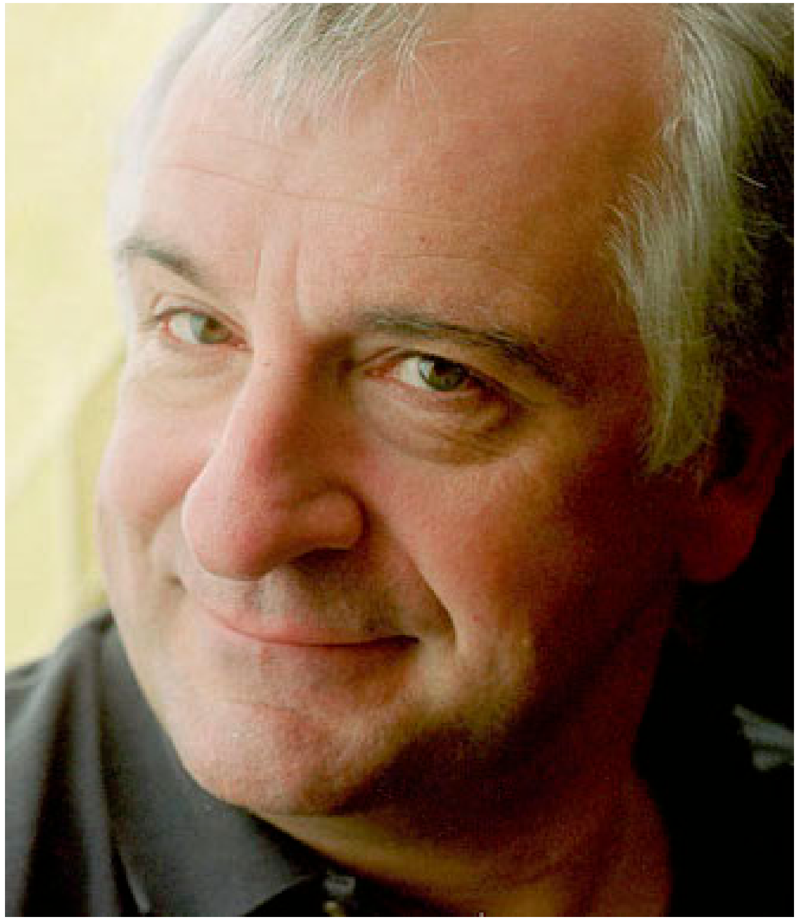
1. What's in the literature
2. The SEP field
3. Elements of best practice
4. Left-shifting – an example
5. Common problems
6. Lifecycle considerations
7. Conclusions and questions

What’s in the Literature

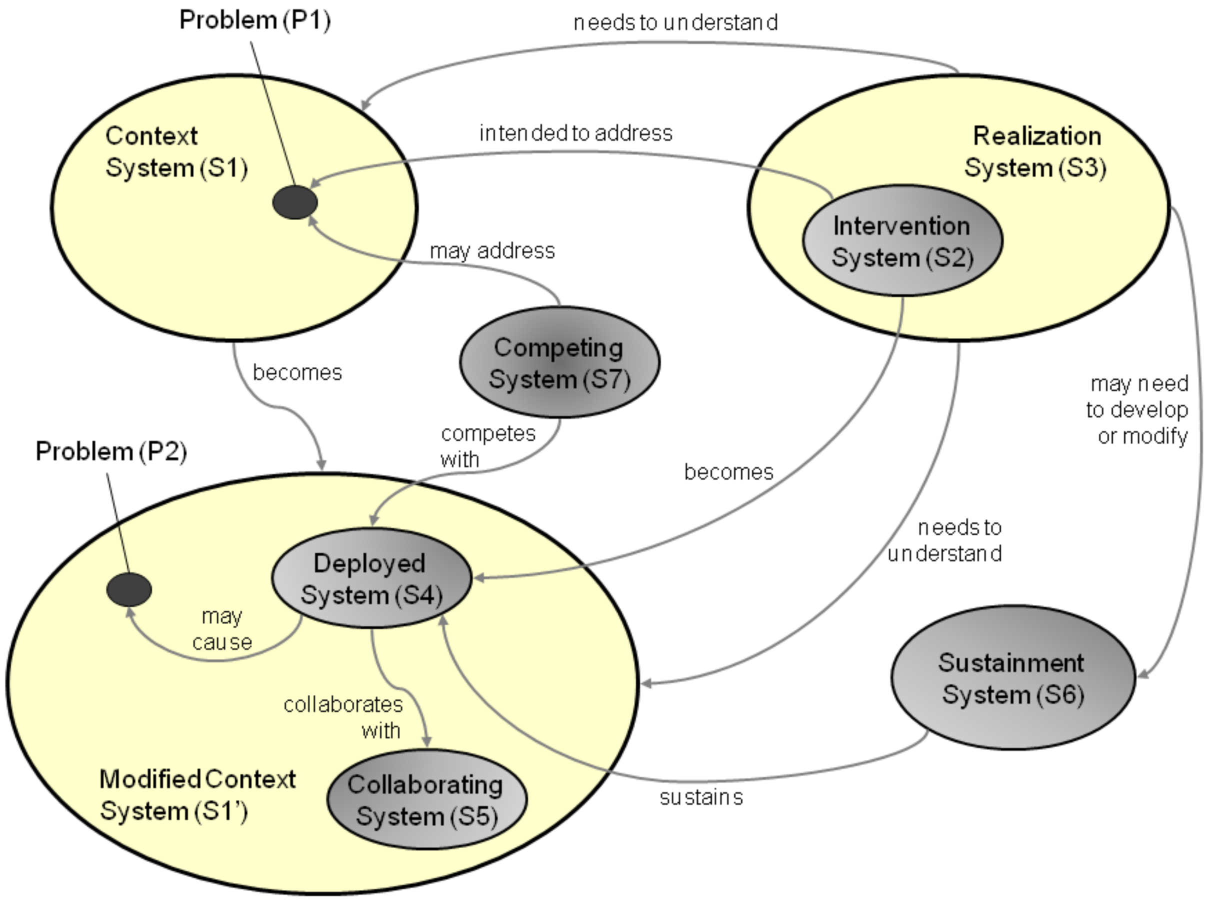


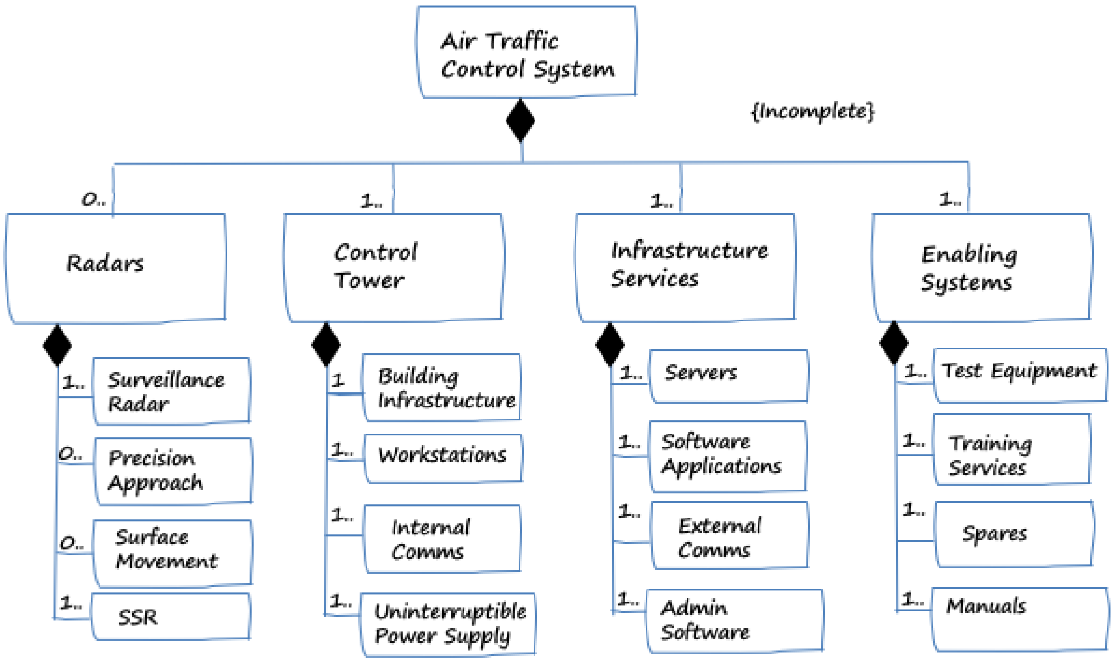


The “Somebody Else’s Problem” field

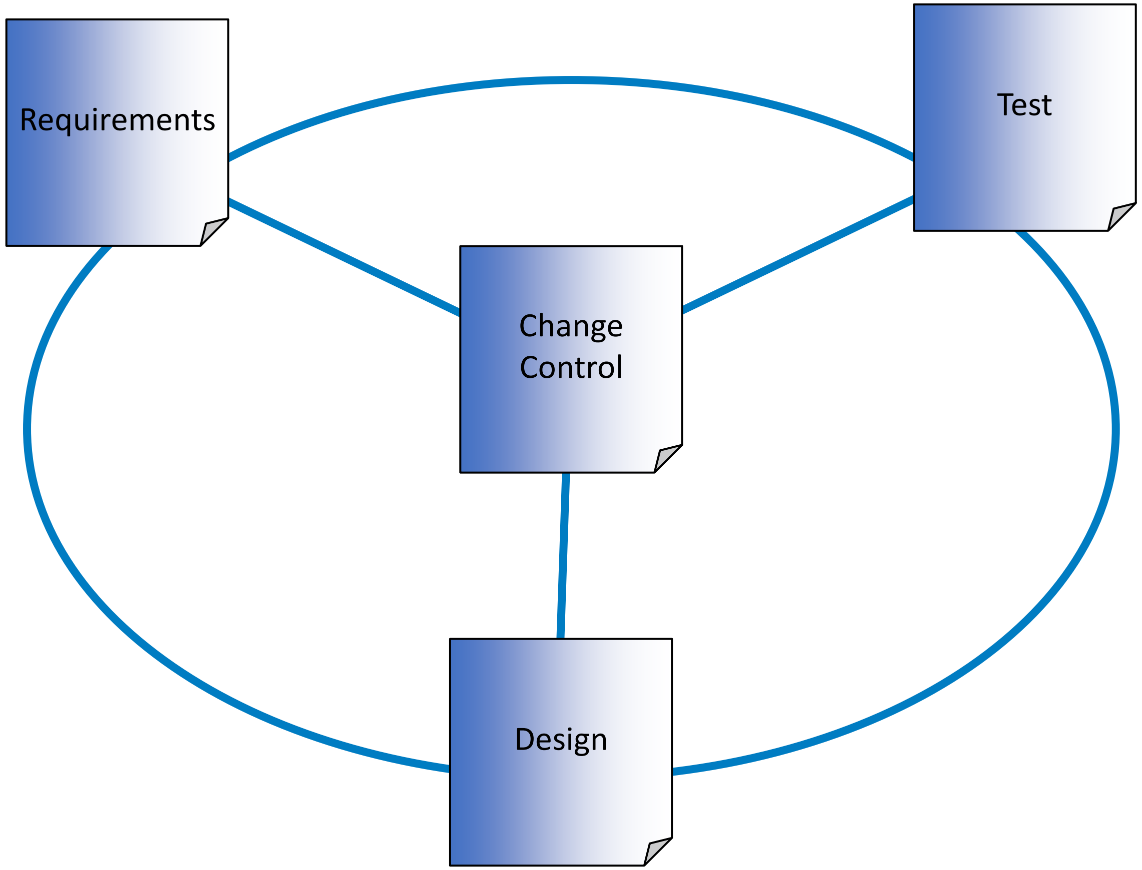


7 Samurai battle the SBS





Why does it matter?

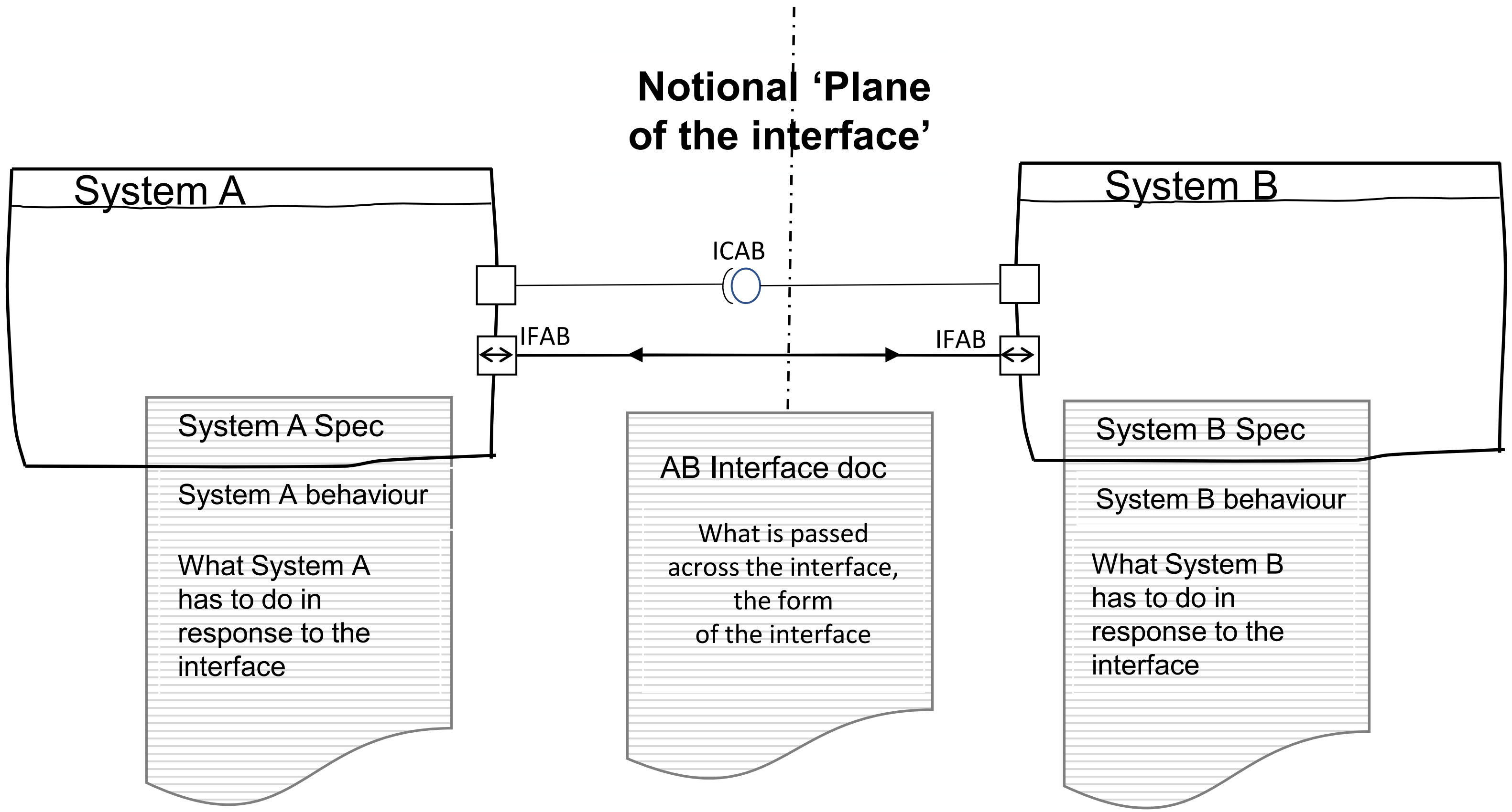


It’s not just software

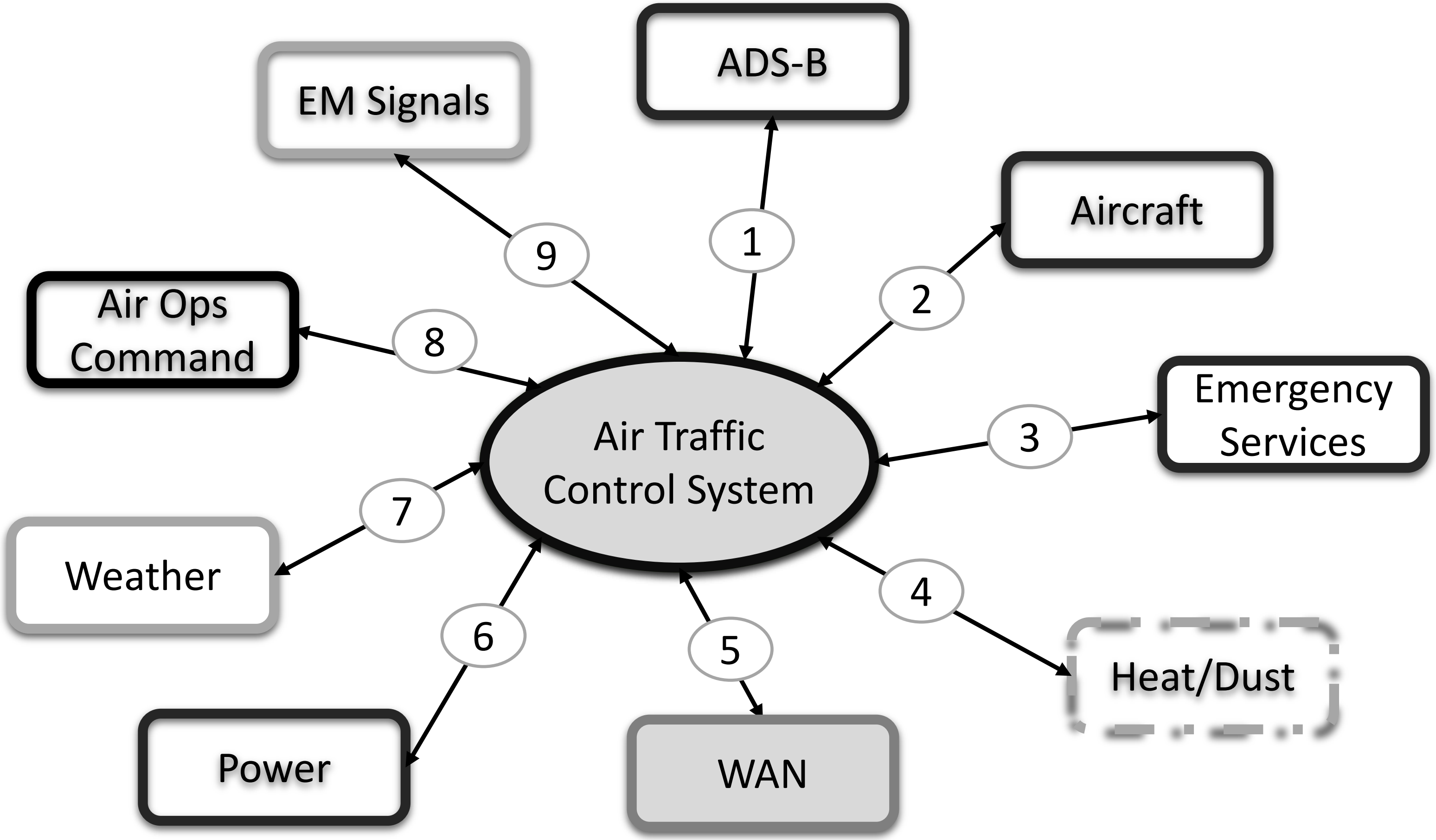
- Electrical voltage + current (+ spikes)
- Vertical forces (time-varying)
- Longitudinal forces due to friction
- Heat
- Flash arcing
- Electromagnetic field flux (+RFI)
- Vibrational forces (resonance?)
- Shock (at joints)

- Moisture & salt deposition
- Carbon deposits, rust, crud

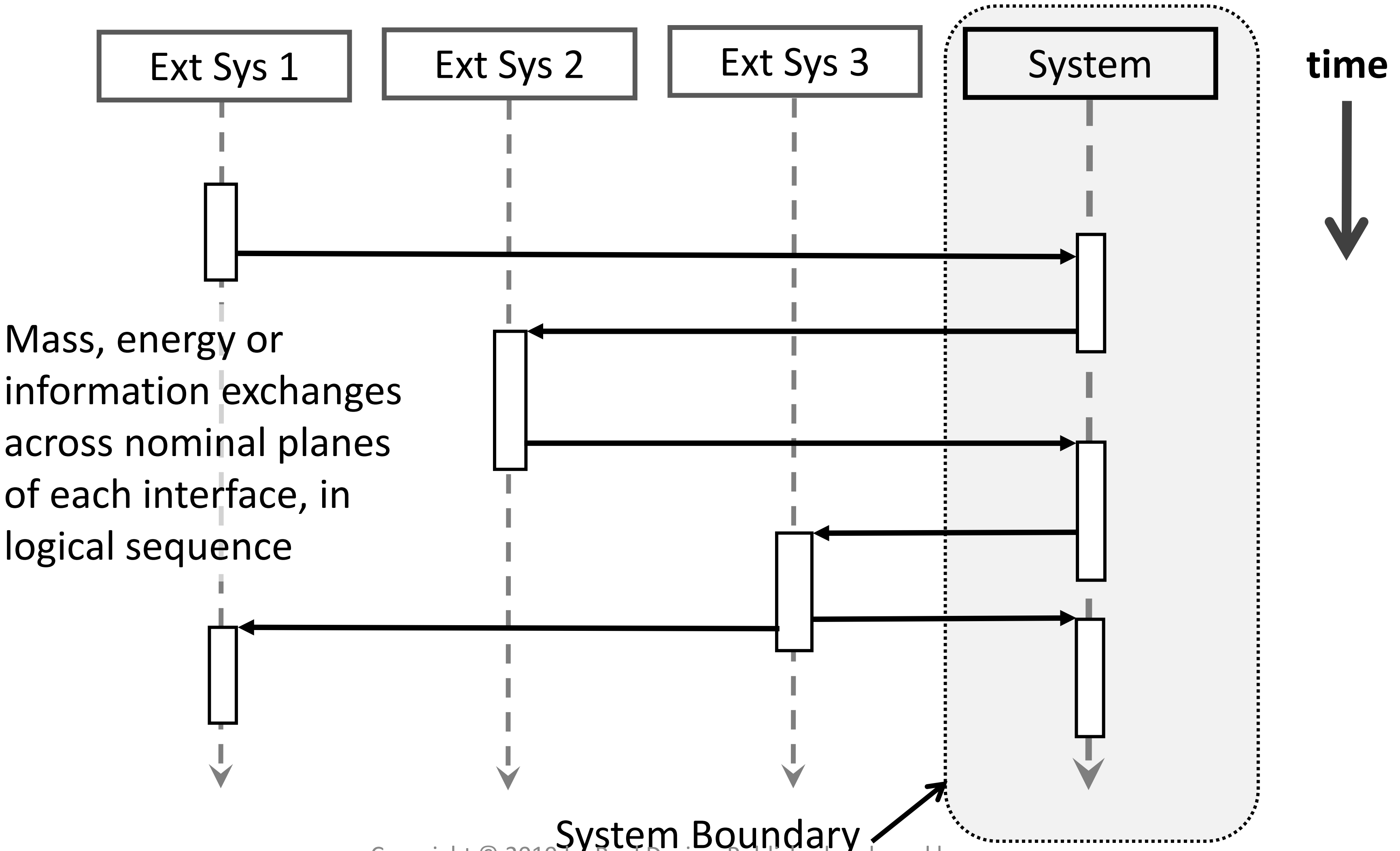
Best Practice 1: the Separation Principle



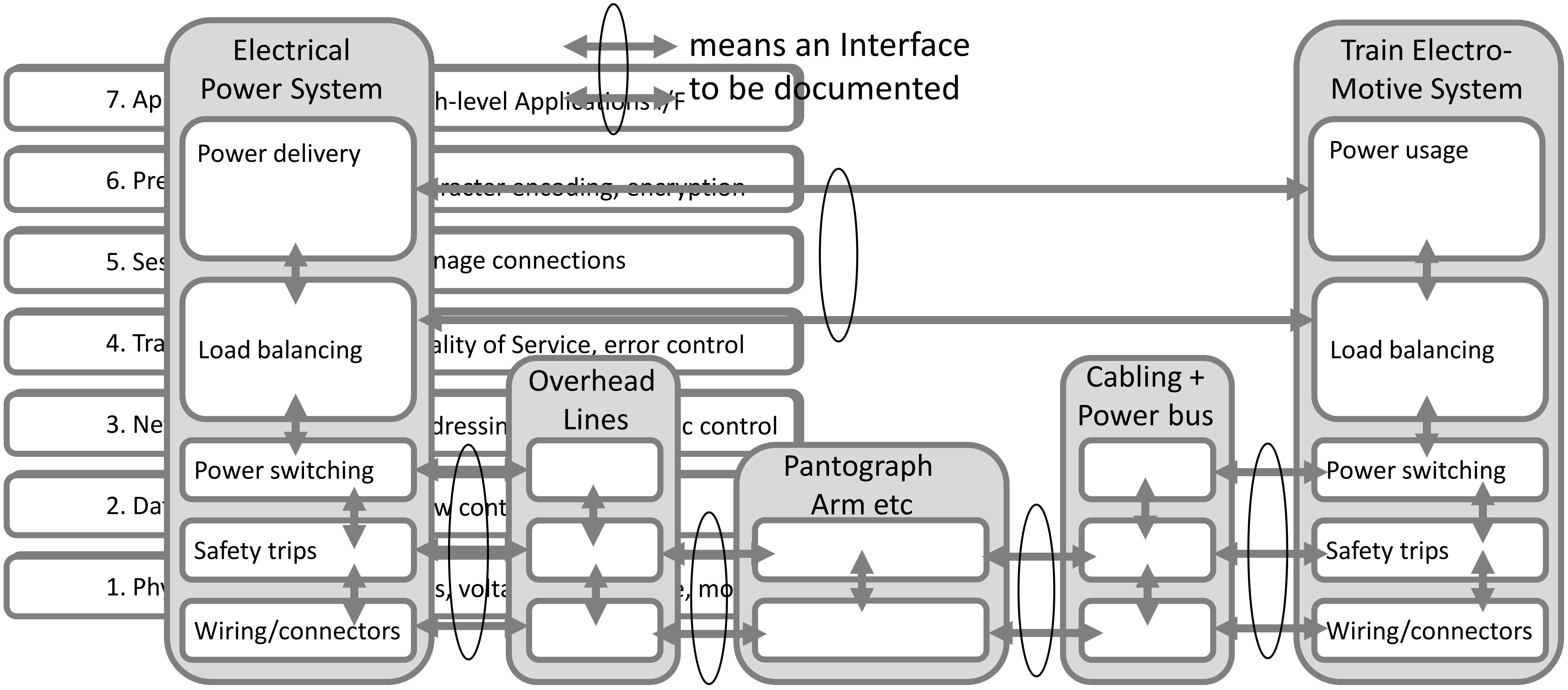
Best Practice 2: the Context Diagram



Best Practice 3: the Sequence Diagram

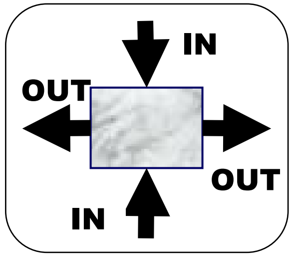


Best Practice 4: layered models as patterns



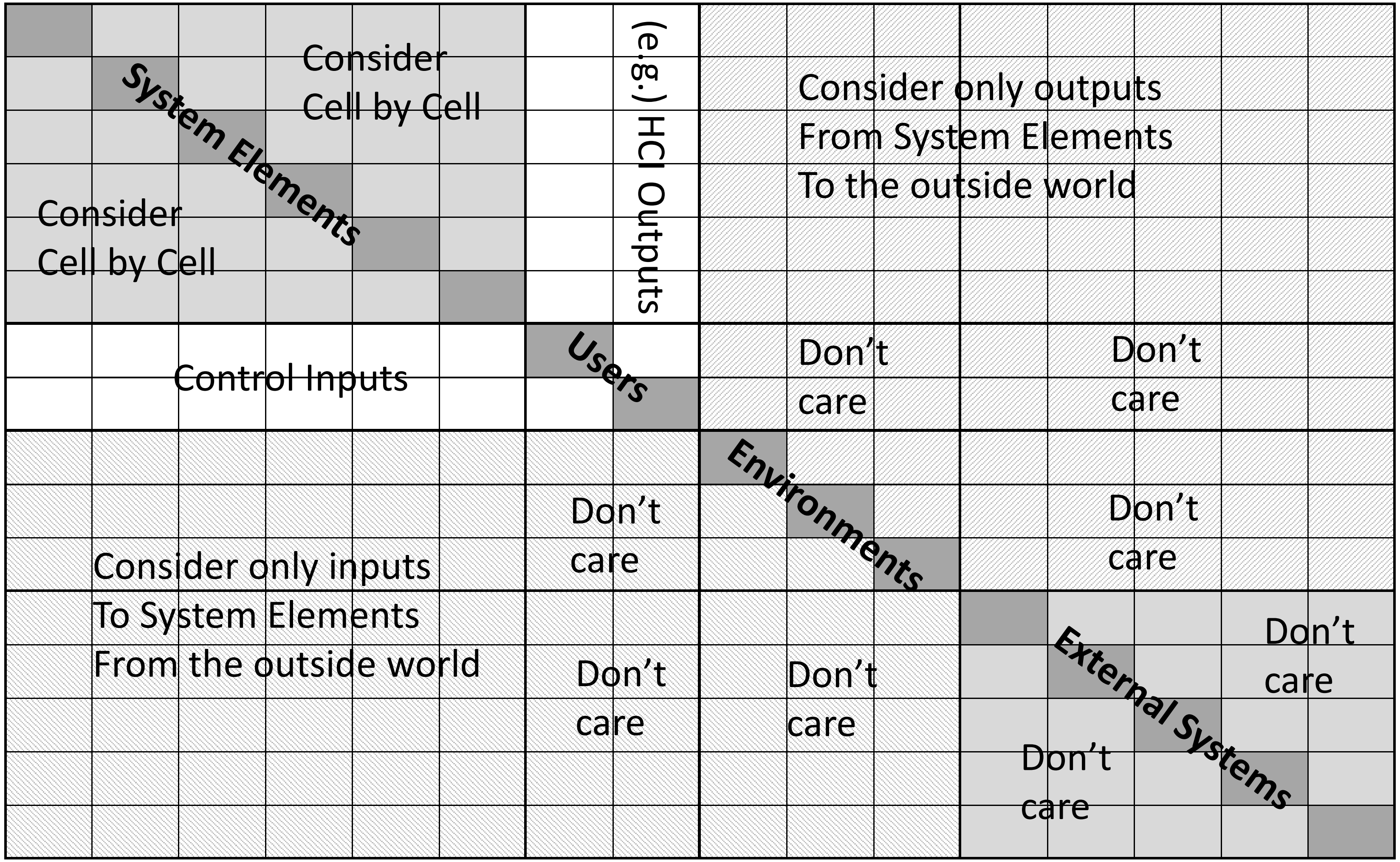
Best practice 5: black box N^2 chart

A -System Of Interest	B requires of A A provides to B	C requires of A A provides to C	D requires of A A provides to D	E requires of A A provides to E
A requires of B B provides to A	B - Users	C requires of B B provides to C	D requires of B B provides to D	E requires of B B provides to E
A requires of C C provides to A	B requires of C C provides to B	C - Physical Environment	D requires of C C provides to D	E requires of C C provides to E
A requires of D D provides to A	B requires of D D provides to B	C requires of D D provides to C	D - External System 1	E requires of D D provides to E
A requires of E E provides to A	B requires of E E provides to B	C requires of E E provides to C	D requires of E E provides to D	E External System 2

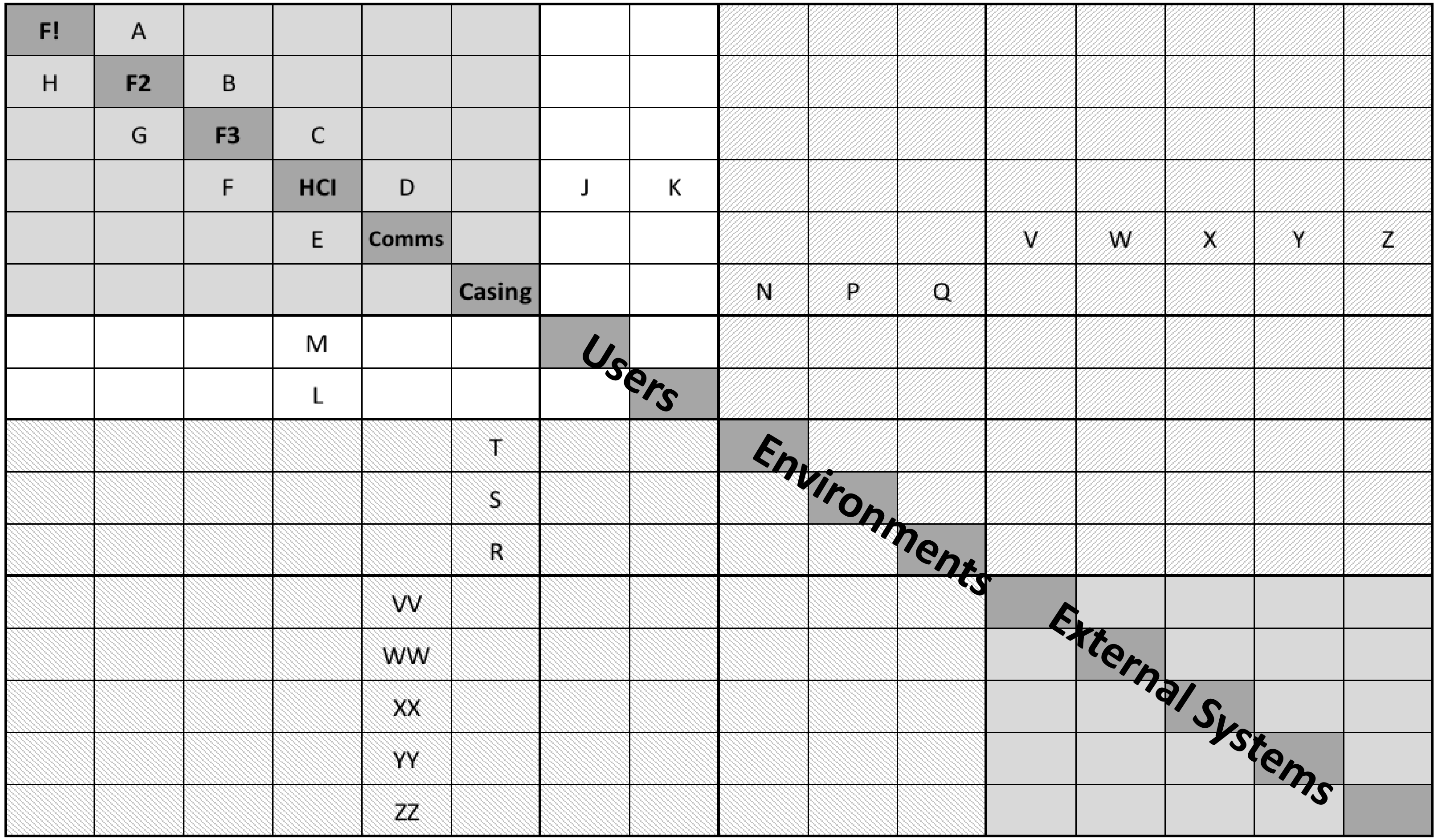


...etc

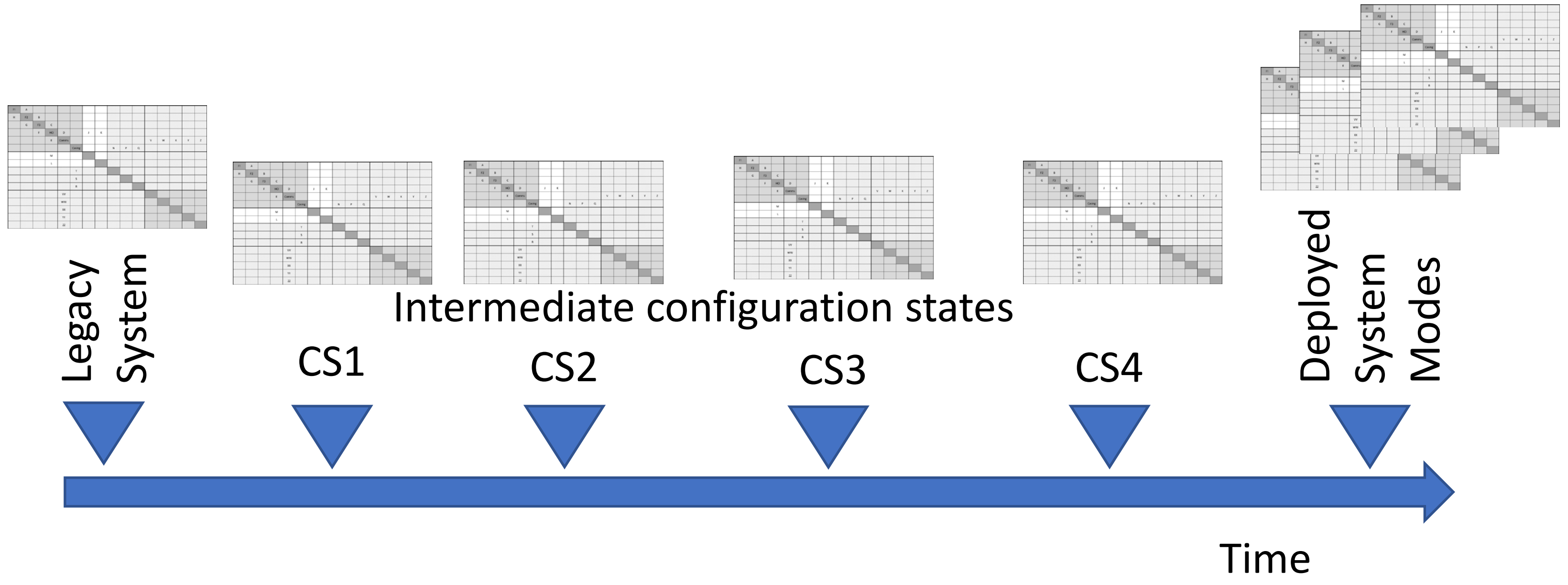
Best practice 6: white box N^2 chart



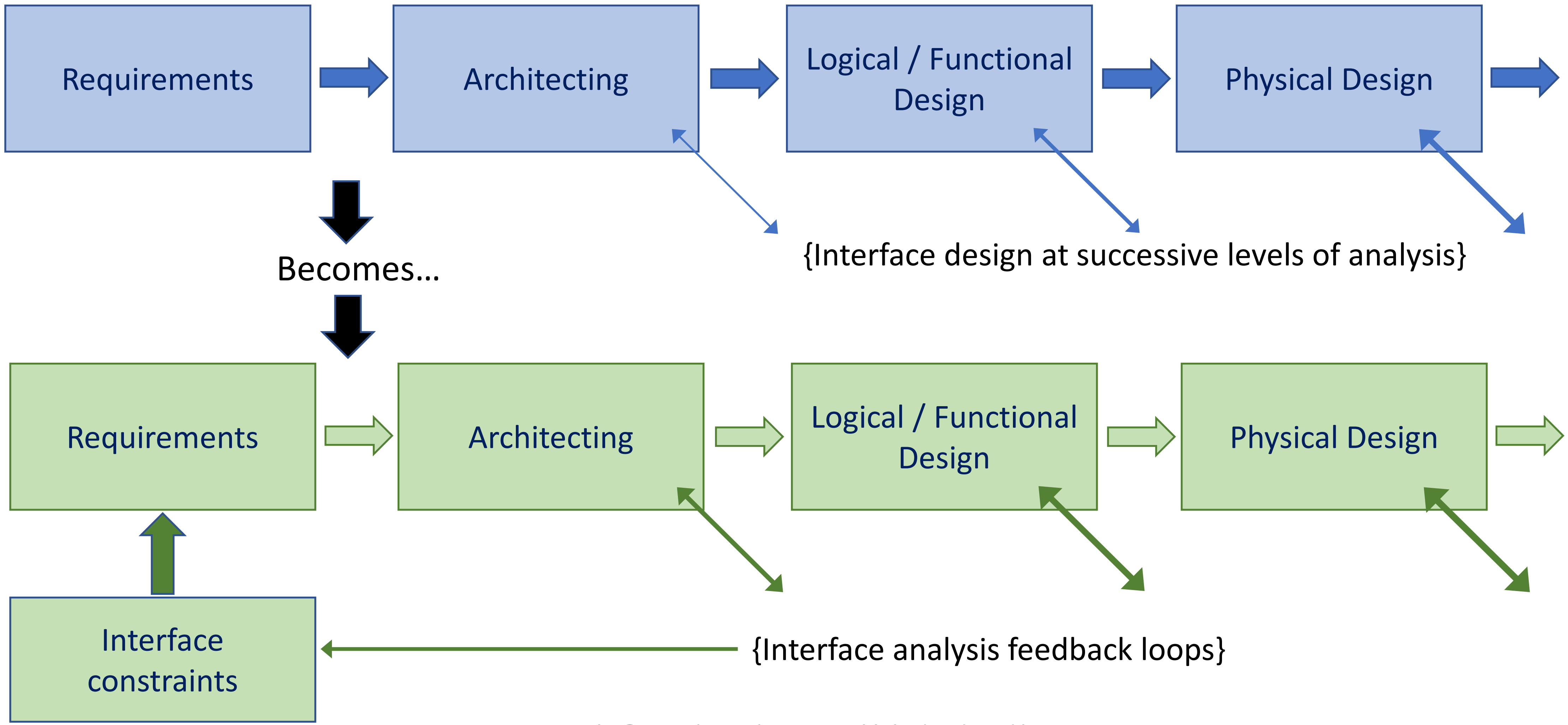
Best practice 7: optimised N^2 chart



Best practice 8: phased implementation N^2



Left-shifting…



Pantograph example again

Electrical voltage + current (+ spikes)

Vertical forces (time-varying)

Longitudinal forces due to friction

Heat

Flash arcing

Electromagnetic field flux (+RFI)

Vibrational forces (resonance?)

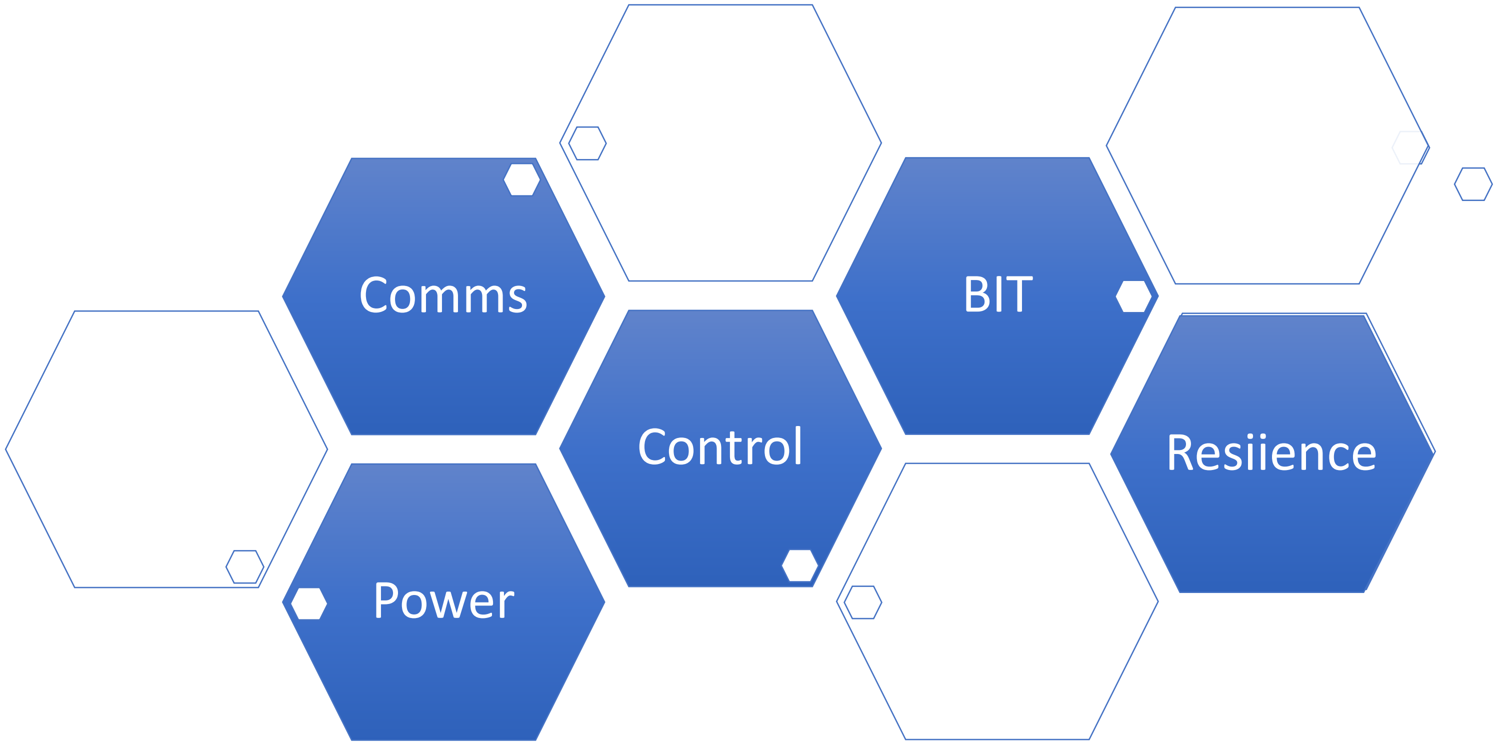
Shock (at joints)

Moisture & salt deposition

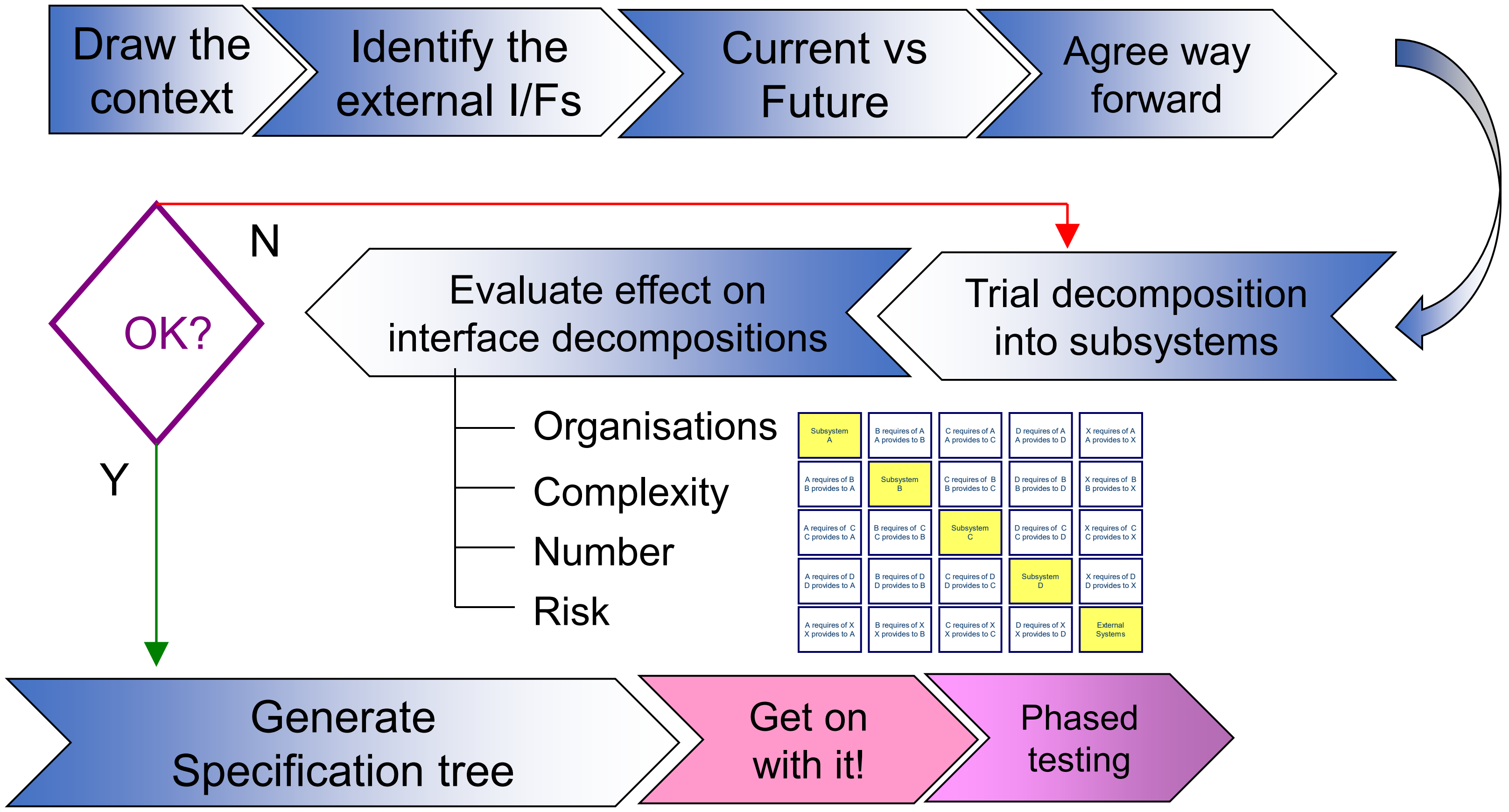
Carbon deposits, rust, crud

The flows across the interface drive extra functional and non-functional requirements on the System Elements at each end

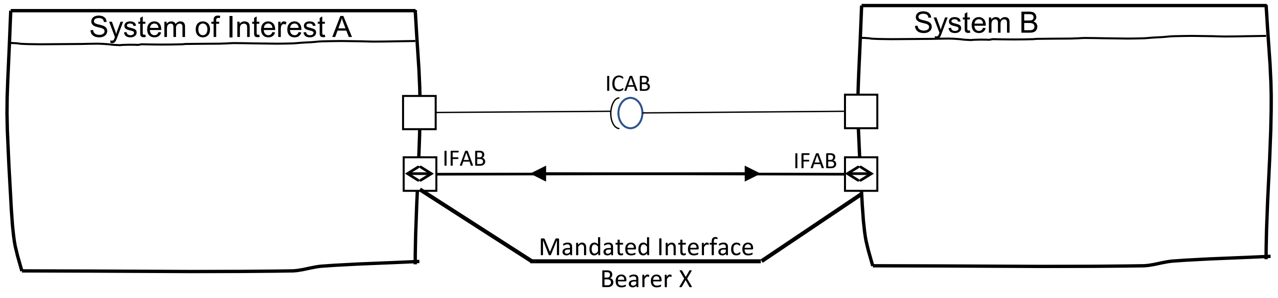
Residual architecting decision patterns for interfaces



Lifecycle of interface-based architecting



The requirement from hell, and future-proofing

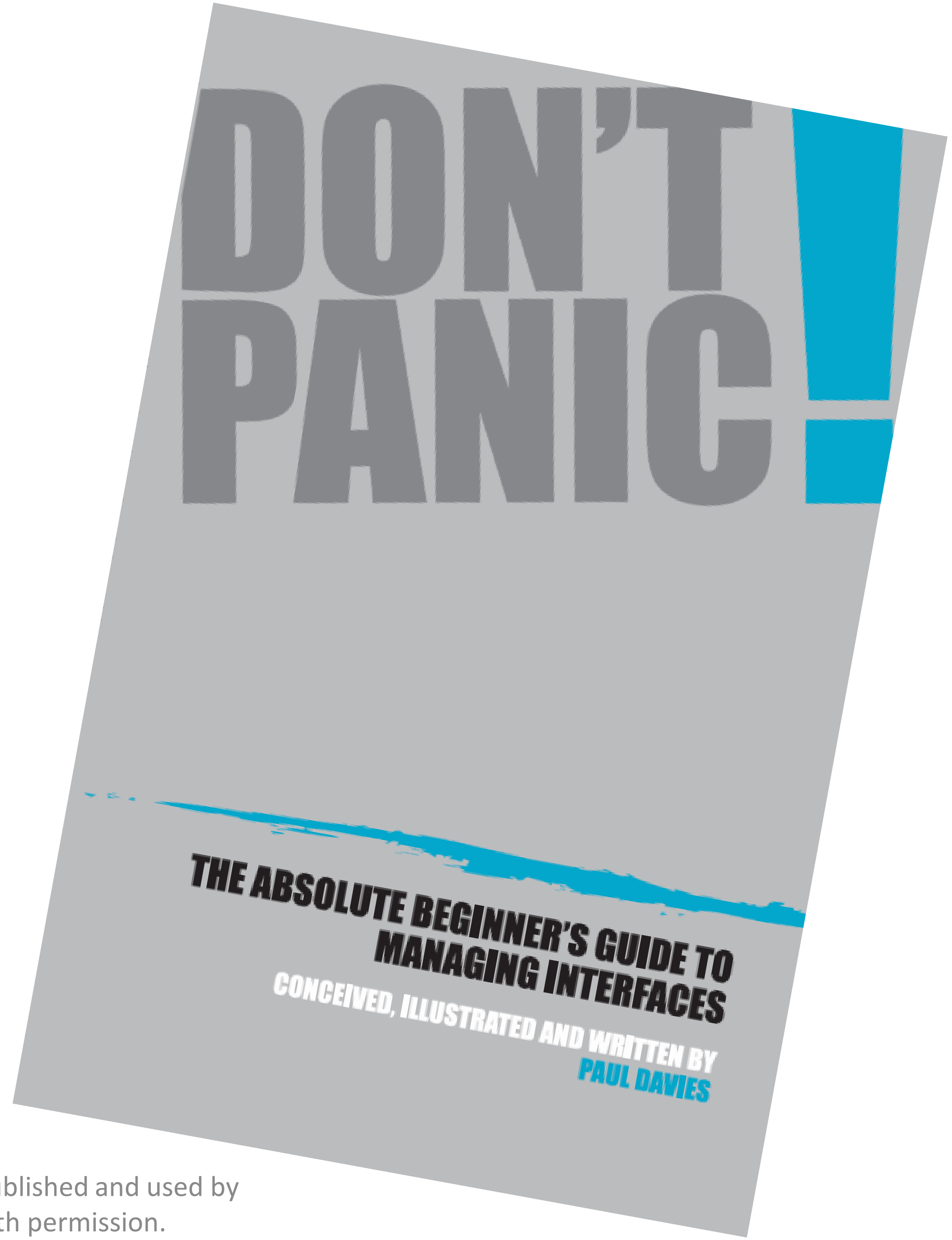


"System A shall interface to System B via bearer X"

1. Bearer X in service; System B to be accepted
2. System A envisaged , with interfaces
3. IDD published for X; IRS created for A<->B
4. System B accepted; ITT for System A uses IRS + IDD
5. System A procured at lower cost & risk
6. Negotiations between suppliers for A and B brokered -> ICD

Conclusions

- We have looked at gaps in the literature, and started to overcome the lack of a lifecycle-oriented view of interface evolution.
- We have outlined some key principles associated with interfaces, and looked at some best practice methods of representing and elaborating them.
- We have stressed the use of interface analysis in architecting Systems throughout their lifecycle.
- We have encouraged engineers to look outside the box.



Interface Management – the Neglected Orphan of Systems Engineering

Paul Davies, paul@thesystemsengineer.uk

Categorisation

- Accessibility: PRACTITIONER
- Application: GOOD PRACTICE
- Topics: Architecture, Systems Engineering Management

Abstract

Every Interface is an opportunity to lose information, time, control and / or money through contention between stakeholders at either end. There are many issues surrounding Interface management, which are relatively unexplored in the engineering literature. Interface management is perceived as a critical skill in the engineering of successful systems, but finding useful material on the subject proves elusive. It is not that there is a gap in the collective Body of Knowledge (BoK) – but there is definitely a gap in the *documented* BoK. This paper explores some of the characteristics of this gap, and strings together some of the key concepts in best practice. Along the way, the differences between best practice for interfaces and best perceived practice for architecting systems are noted, and recommendations for changes in approach are given.

Introduction

Typically, it is an unpopular task on a project to be asked "Just resolve the interfaces"; and whatever effort is allocated, it generally happens too late and is seen to be a root cause of project failures. These are of course sweeping generalisations, yet there is a grain of truth here; and it becomes a vicious circle of blame waiting for the next project to do the same.

Aims of the paper

- To challenge the perception of an engineer as someone who concerns himself (or herself) solely with the realisation of the functionality of their element of the system, to the exclusion of all interactions. Good systems engineers don't do this.
- To remove the excuse "I've never been trained on this, and there is no useful reference material on how to do it."
- To change the way we go about architecting systems; left-shifting the treatment of interfaces rather than leaving it until after the physical design needs integrating.

Literature Survey

We start with a survey of what is documented. Mostly this consists of the usual standards on structuring interface documentation, some process standards, plus there are a few good books on the architecting of systems with at least some relevant content.

Standards

- One of the first standards on Systems Engineering, [IEEE 1220] starts with a System Breakdown Structure (SBS) devoid of interfaces as early as page 3. Admittedly there are better figures, and process requirements to specify interfaces at each level of system decomposition, later in the standard, but in each case the treatment of interfaces is as an adjunct to the act of specification at that level. No “how-to” process detail is offered.
- A more useful standard, [EIA-632] contains a grand total of 7 lines on interface definition, one of which is now generally agreed to be bad practice; plus one table on recommended processes for system decomposition which is almost a copy of the process in [IEEE 1220].
- Further domain-specific standards, [DI-IPSC-81434], [DI-IPSC-81436], [FAA-STD-025e], [NASA 1997] and [NIST 2002] are significantly better, and for some types of interface (mainly software and communications) give good guidance on interface specification content – but still have shortcomings in other engineering fields and domains.
- The INCOSE Systems Engineering Handbook (SEH) [INCOSE TP-2003-002-04], in turn based on ISO15288, is better still, and considers interface analysis as part of the process on system architecture definition. There is another short section on Interface Management as a cross-cutting technology, but detail is light.
- The INCOSE Systems Engineering Body of Knowledge (SEBoK) [SEBoK 2018] has some good content, particularly in the sections on “Synthesizing possible solutions”, “System architecture”, “Logical and Physical architecture model development”, and “System Integration”. There are also several interesting examples and case studies of good and bad practice and outcomes. Taken as a whole, it avoids most of the shortcomings listed under “Gap analysis” below, but it lacks an integrated lifecycle-based treatment of interfaces.
- The INCOSE Competency Frameworks [INCOSE TP-2018-002-01.0] and its 2010 predecessor identify Interface Management as an essential competency in its own right. There are some brief points on good practice, and how to spot it, but no end-to-end narrative.

Books on system architecting

[Hitchins 1992], [Grady 1994] and [Sillitto 2014] all have good treatments on N^2 charts (or “Coupling Matrices” as used by Grady and by the SEH), and on their use in changing or optimising architectures. However, they are quite hard to follow in some cases, and are not fully integrated with other interface concepts described below. All are recommended reading anyway, for the aspiring system architect!

Gap analysis

In summary, the documented body of knowledge is deficient in the following areas:

- It’s all about the “what” must be done, and in what format, but there is not enough useful instruction on the “how”.
- It mostly concerns software and communications interfaces, particularly the Standards.
- It is focused on decomposition of systems into system elements in the strictly functional, then physical order, with interfaces as adjuncts at each level. Very little consideration is given to architecting by minimisation of interface complexity, or to using interface analysis iteratively with other architecting methods.

- Interface analysis at each level of decomposition is treated as a snapshot activity, with no end-to-end timeline of project practices in interface management.

The “Somebody else’s problem” field

Douglas Adams, in his novel “Life, the Universe, and Everything” [Adams 1982] described the Somebody Else’s Problem (SEP) field as “something we can't see, or don't see, or our brain doesn't let us see, because we think that it's somebody else's problem. The brain just edits it out, it's like a blind spot.” Interfaces can easily be subject to the SEP field, as engineers are pre-programmed to worry about internal functionality of a system or system element.

Consider the Seven Samurai model of a system development – see Figure 1, reproduced by kind permission of James Martin [Martin 2004]. It implies the interaction between the system and the problem space, and there are interactions with all other systems depicted, over a timeline. And yet, the temptation for an engineer is to leap straight to solution space, and at least mentally to draw a System Breakdown Structure, see Figure 2, within a few seconds.

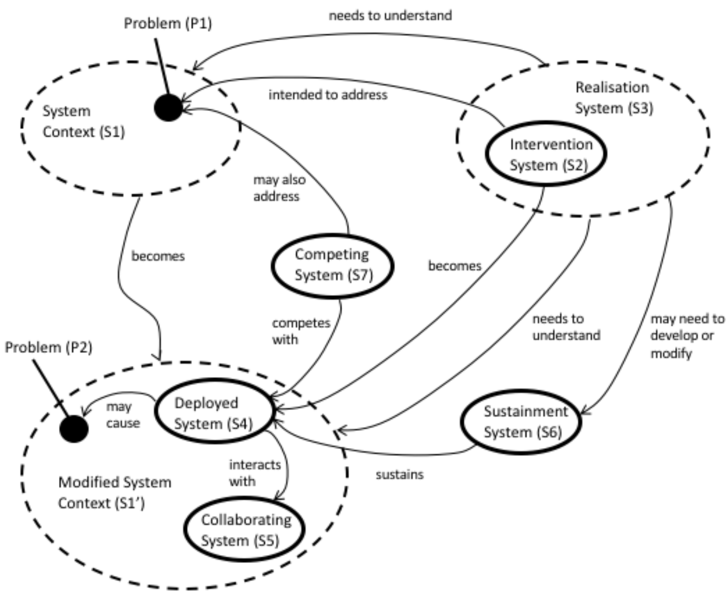


Figure 1 - Seven Samurai

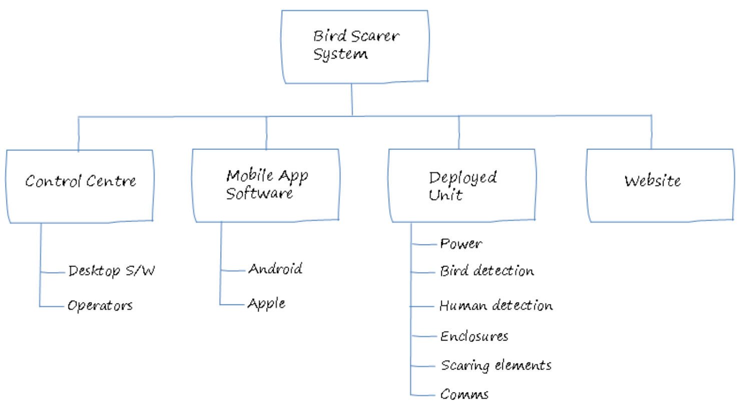


Figure 2 - System Breakdown Structure

Comparing Figure 1 with Figure 2 – where have all the interfaces gone? In the SBS, we cannot see the black-box external interfaces at system level, nor the white-box interfaces between system elements. We have successfully created the SEP field. And now a Work Breakdown Structure (WBS) will be created to match the SBS, again deferring any consideration of interfaces.

Does it really matter? Can’t we allocate responsibility for the interfaces to the engineer responsible for each system element? Here are some reasons why not:

- Every interface connects at least two systems or system elements. In the majority of cases, there is no single span of control over both ends – so the interaction between them needs at least two-party negotiation and agreement.

- Integrating across interfaces takes *longer* than integrating the functionality of a system element. So designing and testing the interfaces has to happen *earlier*.
- As systems are decomposed into system elements, the number of potential interfaces grows much faster than the number of elements. In the SBS example at Figure 2, if each system element is connected to every other, and to 2 external systems, that's potentially 14 interfaces to manage. And there will be many more at the next level of decomposition. The numbers may be an exaggeration, but the principle holds true.
- Traceability – interface requirements may need to trace to the specifications at each end, as well as to the parent system.

Hence the effort needed is seen to outstrip the effort allocated to a single system element, in a non-linear manner. Systems engineers need to focus attention on interfaces, particularly in early planning and estimating stages of projects, to overcome the SEP field.

Elements of best practice

In this section, the key concepts of interface management are briefly described, and logically chained together. Space here does not allow this paper to provide full explanations, and architecting involving interfaces does take significant time to master. It is hoped that the systems engineers will mostly be familiar with all these concepts, but repeated deliveries by the author of a tutorial on this topic would suggest that this may not be the case. If the reader is indeed unfamiliar, try reading [Davies2019] for help. See Figure 3 for the key concepts.

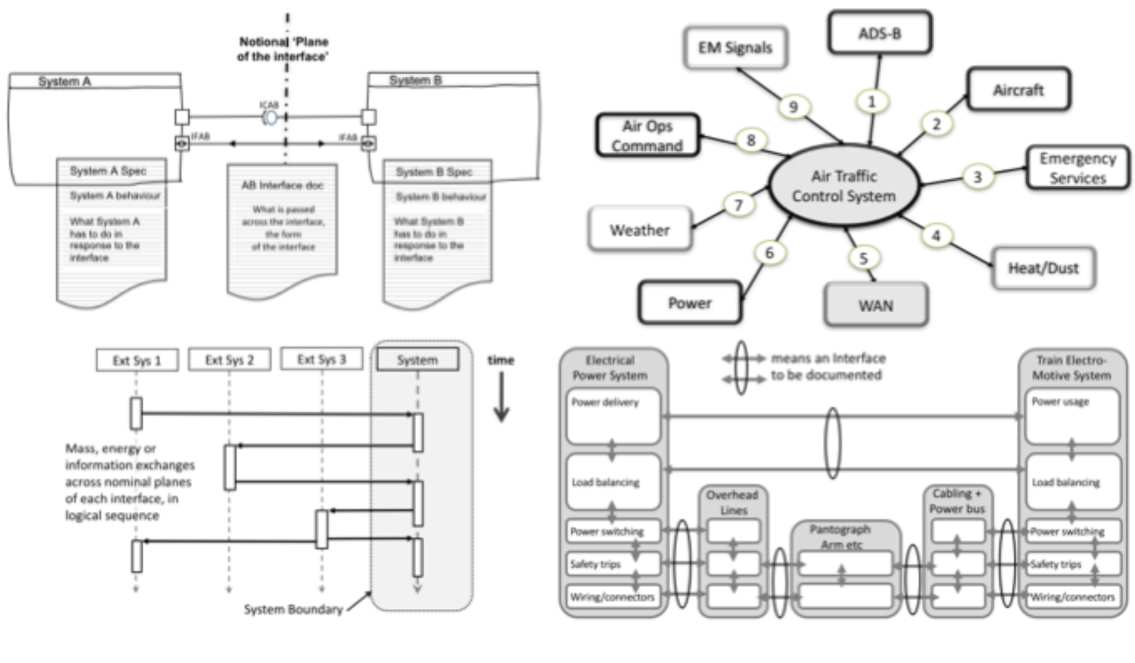


Figure 3 - Useful concepts in Interface Management: Separation principle (top left), Context diagram (top right), Sequence diagram (bottom left), layered instantiation of interfaces (bottom right).

The *separation principle* simply says that interface specifications should not contain descriptions of interaction functionality. They should go in the functional specifications of the interacting entities; the bounds of exchange sequences could go in a higher-order (e.g. containing system) specification.

Black-box and white-box models are essential items in any systems engineer's armoury. Black box - the view of the system functions and interaction observable at the system boundary, without knowing anything about its internals. White box – extending the model to include the system internals and all their interactions.

Context diagrams are a convenient representation of a black box model, showing a system boundary separating what is inside the scope of the system from what is outside; the external systems, actors and environments with which it interacts; and enumerating those interactions.

Scenarios & sequence diagrams are helpful pictures in eliciting interface requirements, by turning stakeholder descriptions of interaction sequences into pictorial sequences of exchange of mass, energy and information. First at black box level, then extending to white box.

Layered models of interfaces, for example the "OSI 7-layer" model for systems interconnection, is a useful metaphor for dealing with the migration from black box to white box level. System-to-system interaction can be represented as an interface at the "application" layer, which can later be instantiated by 'interactions" downwards into the system element hierarchy and then between the various white box elements; even when the interaction is not just software or communications.

The final key weapon in the systems engineer's armoury is the *N^2 chart* (see Figure 4). This paper does not provide full explanations – see [Davies 2019], or indeed [SEBoK 2018], [Sillitto 2014], [INCOSE UK ω2 2012] or [Grady 1994] which all give at least partial coverage.

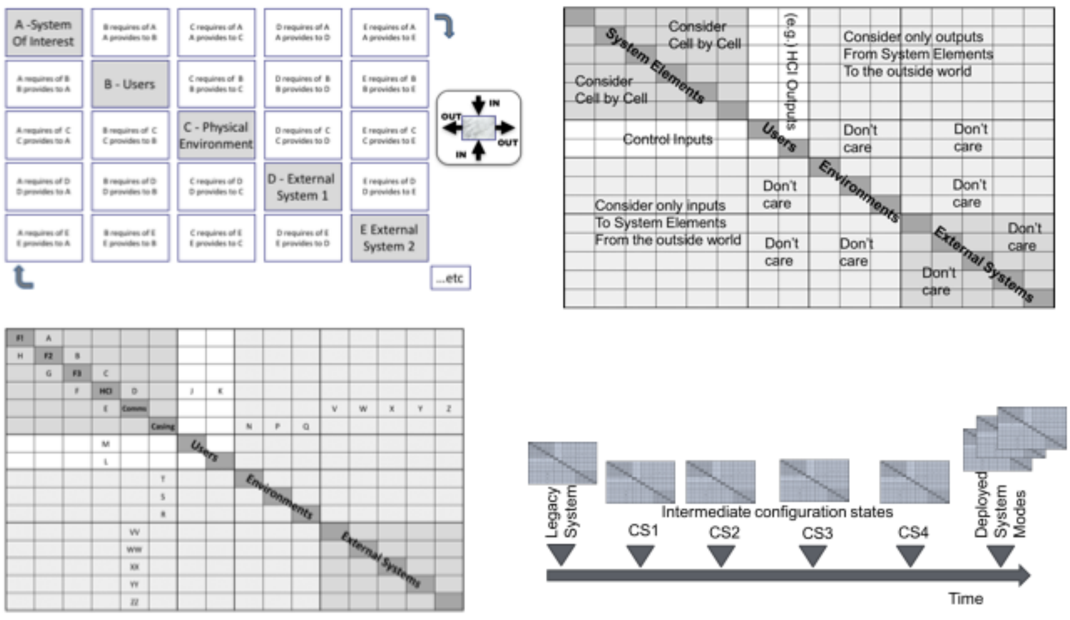


Figure 4 - Uses of N-squared charts in interface management: Black box (top left), white box (top right), optimal architecture (bottom left), phased timeline (bottom right).

N^2 charts may be used successively at black box level, white box level, and iteratively with system architecting to minimise number and complexity of interfaces in trial decompositions. Some of the texts quoted call this "reorganisation of coupling matrices", but those treatments do not attempt to re-define system elements to *change* the entries in the N^2 charts. Finally in Figure 4, we note that there is not just one N^2 chart in the life of a project, there are many: for as-is and to-be systems, phased integration setups, intermediate delivery configuration states, and configurations to support deployment, maintenance, in-service test, replacement and upgrade.

Left-shifting in architecting systems

The use of interface analysis as an up-front tool in architecting, rather than as a capture mechanism for managing a design that has already been decomposed, is illustrated with several recurrent architecting problems requiring this modified approach. Thus we achieve better integration of logical and physical architecture.

Example - Overhead Line Electrification (OLE) in rail

Consider a pantograph arm mounted on top of an electric train (Figure 5). Its mechanical interface with the overhead line electrification cable may be moving at more than 100mph, sliding from side to side of the pantograph arm, and electrical connectivity may be interrupted for short periods of time. And yet we can still think of it as a single plane of the interface, across which a number of mass, energy and information flows take place.

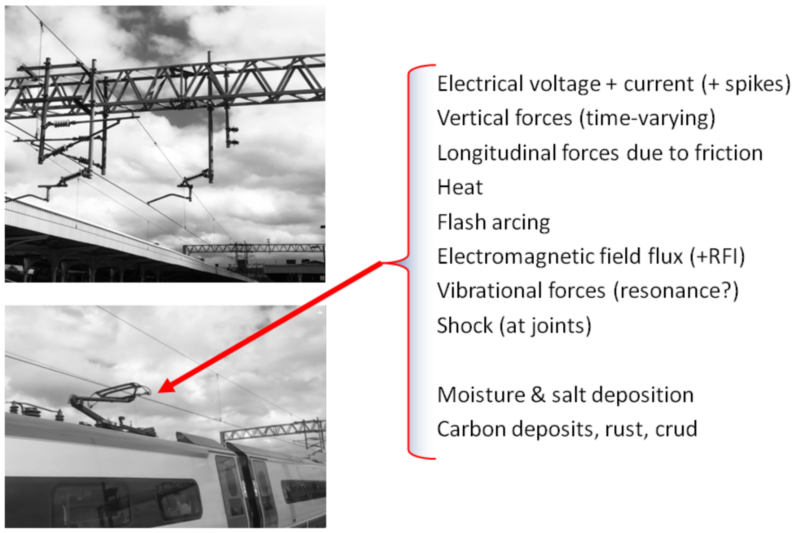


Figure 5 - Overhead Line Electrification

Some of these are interfaces with other systems or system elements; some are interfaces with the environment. However, in each case our system (the train) must do something in response to what is happening across the interface.

If, at the requirements and architecting stages, we were to focus only on the *intended* usage of the interface (electrical energy transfer), we would miss all the (undesirable) aspects of the other issues in the interaction. By the time these were considered, we might already have made design choices which made the handling of the undesirable characteristics impossible, over-expensive or unmaintainable. So, by consideration of the effects of the interface at the physical instantiation, we collect a set of additional system requirements and design drivers, in a timely manner. Incidentally, in an exercise left to the reader, this is also an excellent example of the use of a layered model of interfaces in resolving the transfer of electrical power from generation capability to electromotive capability of the train.

Common residual architecting problems involving interfaces

There are some recurring patterns in system design that impact on architecting decisions about interfaces, all adding to the case for including interface issues much earlier than in common practice.

Power – Assuming there is a single external source of power for the system, whether it be generator, mains, or aircraft high-voltage DC, how best to supply power to our system elements? Should each unit do its own conversion from the external source (so multiple external interfaces), or have a single power supply unit downconverting to the voltages and currents required by all the other units (multiple internal interfaces)? The former may be harder to organise, and carries potential safety issues. The latter is "easier", in the sense that the multiple Interfaces are under single span of control, but gives a single point of failure.

Communications – Likewise, is each system element going to communicate with external systems, or will there be a single "concentrator" system element or nodal point? The former allows design teams to proceed independently – perhaps faster to implement, but potentially leading to integration headaches. The latter allows a global overview of all the external interfaces, but centralises a potentially heavy workload.

Control – For a system with diverse functionality in response to either external conditions, or to operator input, control of the behaviour of each system element can be centralised or distributed. If distributed, it can be difficult to guarantee coherence of the integrated system. If centralised, there is still a choice between high-level and low-level control signals or messages. The former ("Do this, you work out how") makes for a simpler interface but a more complex design task and integration sequence. The latter ("Do exactly this, I've worked out how, here are the control signals") makes for more complex interfaces, perhaps a simpler integration sequence, and a higher centralised workload. This is a more acute problem if the system has multiple states and modes.

Built-In Test (BIT) – This is exactly analogous to the control problem above. "Test yourself, tell me whether or not you're OK" (simple interfaces, distributed design, risky integration) or "Here are the test signals, I'll interpret the results" (complex interfaces, single complex system element design)?

Environmental and mechanical resilience – For a system with groups of co-located system elements exposed to a harsh environment, should we design and test each system element to survive that environment? Or should we design a protective casing that insulates the contained system elements from that environment? If there are existing designs for the former architecture that meet, or can be modified to meet, the environmental requirements, that is probably simplest. However, if this is not the case, or we wish to reduce costs by using commercial equipment not designed to resist the environment, then the latter is probably best. Beware, however, that no protective casing is perfect, particularly for shock and vibration. We will have a number of derived environment and mechanical interfaces for each contained system element, which may be difficult to calculate and test.

These are all examples where the impact of the interfaces needs to be treated as a major driver in architecting, rather than as a follow-on activity to functional and physical decomposition.

Lifecycle considerations

Having accepted the principle of including black-box and white-box interfaces in system architecting choices, a flow diagram for the process is suggested at Figure 6.

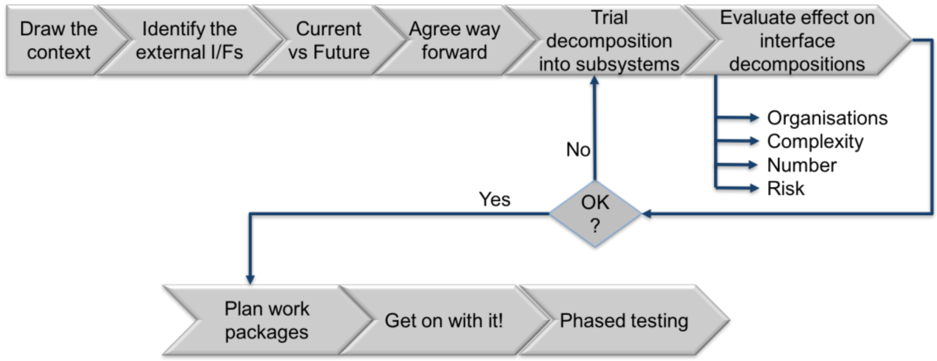


Figure 6 - Lifecycle of interface-based architecting

There is a key architecting loop clearly shown. Candidate system decompositions are evaluated based on the organisations involved (and perceptions of their willingness to negotiate and adapt to a satisfactory interface agreement), and the number, complexity and risk of the interfaces derived. The fundamental change in philosophy from most texts is that the architecting stage is not concluded until the interfaces are deemed satisfactory. Note again the bottom right-hand quadrant of Figure 4, and its supporting description: the final system configuration is not the only set of interfaces to be analysed and included in the architecting loop. Integration and test setups, intermediate configuration states, and considerations from the Deployment and Support Concepts [INCOSE TP-2003-002-04] all affect the architecture acceptability.

Lastly, we look at future-proofing of interfaces to and from systems yet to be implemented, or legacy systems not under client control.

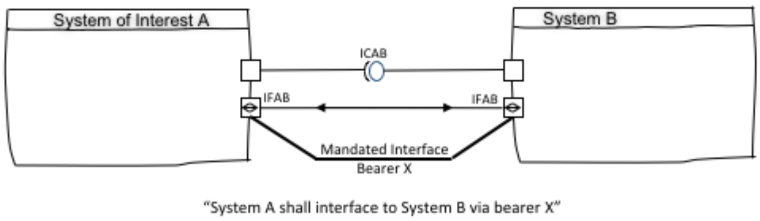


Figure 7 - A typical interface requirement

This is a requirement with a huge hidden risk attached, and this author has suffered from it on numerous occasions. What if the interface to Bearer X is unpublished or immutable? What if System B has a proprietary interface, and its design authority refuses to cooperate? If the acquirer of System A has no authority over the suppliers of Bearer X or System B, this requirement is asking the System A supplier to sign a blank cheque. There is a recommended method for dealing with this situation given in [Davies 2019], based on foreseeing the future requirement and insisting on at least draft Interface Requirement Specifications at the time of entry into operations of X and B; or at least agreed between the acquirer and the owner-operators of X and B.

There may be no truly optimal architecture taking all the foregoing aspects into account, but using the principles outlined, we should at least improve on common practice. Nobody said it would be easy.

Conclusions

- The importance of looking beyond system element functionality to include interfaces has been outlined. Those interfaces need to be resolved: lift up your head, pick up the phone, and resolve them!
- You are now armed with suitable models to deal with difficult interface scenarios, covering the whole lifecycle of both the system and its interfaces.
- Arguments have been presented for left-shifting the treatment of interfaces in architecting.

References

[Adams 1982] Adams, Douglas, "Life, the universe, and everything", Pan Books, 1982

[Davies 2019] Davies P.R., "Don't Panic! The Absolute Beginner's Guide to Interface Management", INCOSE UK, 2019

[DI-IPSC-81434] US Department of Defense 'Data Item Description: Interface Requirements Specification', US DoD, Revision A, 1999

[DI-IPSC-81436] US Department of Defense 'Data Item Description: Interface Design Description', US DoD, Revision A, 1999

[EIA-632] 'EIA-632: Processes for Engineering a System', Electronic Industries Alliance, Philadelphia, PA, USA; 2003

[FAA-STD-025e] "Federal Aviation Administration Standard: Preparation of Interface Documentation", US Department of Transportation, 2002

[Grady 1994] "Systems Integration", Boca Raton, FL, USA, CRC Press Inc., 1994

[Hitchins 1992] Hitchins, D.K. 'Putting Systems to Work', John Wiley & Sons, 1992

[IEEE 1220] 'IEEE 1220-2005: Standard for Application and Management of the Systems Engineering Process', Institute of Electrical and Electronic Engineers, 2005 (superseded[?] by ISO 24748)

[INCOSE TP-2003-002-04] INCOSE Systems Engineering Handbook, Fourth Edition, 2015

[INCOSE TP-2018-002-01.0] INCOSE SE Competency Framework, Issue 01 2018; a revised edition superseding Issue 03 2010 [the version originated by the UK Chapter]

[INCOSE UK ω2 2012] INCOSE UK Omega 2 Guide 'N-Squared: brief guide', Issue 1.0, 2012 – available online to INCOSE UK members only.

[Martin 2004] Martin, J. 'The seven Samurai of systems engineering', Proceedings of the INCOSE International Symposium, 2004

[NASA 1997] NASA Reference Publication 1370 'Training Manual for Elements of Interface Definition and Control', 1997

[NIST 2002] National Institute of Standards and Technology (NIST) Technical Note 1447 'A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts', 2002

[SEBoK 2018] "Guide to the Systems Engineering Body of Knowledge (SEBoK) Version 1.9.1 [online]. Available at sebokwiki.org, accessed 21st May 2019.

[Sillitto 2014] Sillitto, H. 'Architecting Systems: Concepts, Principles and Practice', College Publications, 2014