다시 음미해보는 시스템엔지니어링 원칙
Systems Engineering Principles Revisited

한명덕
Myeong-Deok Han
육군3사관학교 신소재시스템학과

ABSTRACT

After attending the special lecture by Halligan on "The Principles of Systems Engineering" at the 2002 KCOSE workshop, the author tried to collect similar SE principles scanning throughout several SE text books and internet sources. During this process it is found that INCOSE once established a SE-Principles WG and tried to collect SE-principles. They tried to make distinctions among pragmatic, mathematical, and philosophical principles. The result of this effort to collect various SE-principles showed that, including the INCOSE SE-Principles WG, most authors seem only succeeded in generating the pragmatic SE-principles but failed in both mathematical and philosophical SE-principles.

1. 서 론

2002년 KCOSE 워크샵에서 Halligan은 특별 강연을 통하여 시스템엔지니어링의 20 가지 원칙을 제시한 바 있다.[1] 본 저자는 이를 계기로 유사한 시스템엔지니어링 원칙들이 다른 시스템엔지니어링 전문가들에 의하여 어떻게 제시되고 있는지 관심을 갖게되고 이를 수집하여 상호 비교하여보고 싶은 생각을 갖게 되었다.

시스템엔지니어링 원칙에 관한 수집 작업은 주변에 가까이 있는 시스템엔지니어링 교과서 및 핸드북 몇 권으로부터 시작하였다. 이어서 인터넷에 올라와 있는 시스템엔지니어링 원칙들을 찾기 시작하면서 INCOSE에서 유사한 작업을 시도한 적이 있음을 알게 되었다. 이리하여 INCOSE의 SE Principles 워킹그룹 활동에 관한 자료를 수집하였고, 이어서 영국의 다른 기관에서 역시 시스템엔지니어링 원칙을 정리하려고 노력한 결과를 수집하게 되었다.

이렇게 수집된 자료들은 서로 비교한 결과, 시스템엔지니어링 원칙에 대하여 몇 가지 개념적인 구별이 존재할 수 있음을 알게 되었다. 예를 들면 INCOSE의 SE Principles 워킹그룹 같은 경우에는 실용적 원칙, 수학적 원칙, 철학적 원칙을 구별하려고 노력하였고,[2] Rechtin 같은 사람은 서술적 원칙과 처방적 원칙을 구별하였음을 알 수 있다.[3]

이 논문에서는 원저자들의 뜻을 명확하게 전달하기 위하여 수집된 원칙들은 굳이 번역하지 않고 원문대로 옮겨 적고 단지 편의상 번호들은 새롭게 부여하기로 한다. 다만, Rechtin의 경우는 제시된 원칙이 너무 많아서 일일이 제시하는 것은 생략하기로 한다. 수집된 원칙들을 비교 검토한 결과는 결론으로 제시하겠지만 간단히 말하자면 이들 원칙은 경험에 입각한 실용적 권고사항들이라고 보는 것이 타당할 것이다.

2. Halligan의 원칙 (2002년)

2002년 11월 한국 시스템엔지니어링 협회는 서울대학교에서 시스템엔지니어링 워크샵을 개최하였다. 마침 5일 과정의 시스템엔지니어링 강좌를 진행하고 있던 Halligan은 이 워크샵에 특별 강연을 담당하여 시스템엔지니어링의 원칙에 관한 강연을 실시하였다.

Halligan이 "Systems Engineering Principles - Essence and Implementation" 제목의 이 강연에서 제시한 시스템엔지니어링의 원칙은 다음과 같은 20 가지 원칙이었다.[4]

1. Capture and understand requirements, measures of effectiveness (MOEs), goals and value of outcomes before committing to the solution to a problem.
2. Ensure that the requirements are consistent with what is believed will be possible in solutions at the time of physical implementation, i.e., are feasible.
3. Treat as goals desired characteristics that may not be feasible.
4. Define system requirements, measures of effectiveness (MOEs), goals and solutions having regard to the whole of the (remaining) life cycle of the system of interest.
5. Design a solution by dividing the big problem into a set of individually well defined smaller problems, i.e., by defining the required characteristics of each element of the solution (including both product and process elements).
6. Use sequential development (waterfall, grand design, "big bang", etc.) for design where requirements (etc) are well defined and stable, and solutions are relatively simple or well understood.
7. Use incremental development where requirements (etc) are well defined and stable, but solutions have risk due to technology and/or due to complexity.
8. Use evolutionary development where requirements (etc) are as well defined as is possible in the circumstances, but remain inadequately defined, or are subject to change.
9. Use a stage-based, stage gate, risk-driven style of

development as an overall strategy for system development.

10. All of the systems engineering process elements exist within the context of sequential, incremental, evolutionary and risk-driven style of development.

Design the development process to match the nature of the problem, using the SE process elements as building blocks.

11. Maintain a distinction between the statement of the problem to be solved and the description of the solution to that problem, for the system of interest and for each element of the selected solution.

12. Baseline the statement of the problem to be solved and the description of the solution to that problem.

Control changes to requirements (etc) and to design, maintaining traceability to the applicable baseline.

13. Identify and develop solution alternatives that are both feasible (i.e. can meet requirements) and are potentially the most effective.

NOTE : MOEs could include development cost, unit cost of production, time-to-market and other measures unrelated to capability of the product when used.

14. Develop solutions for relevant enabling systems concurrently, and in balance, with the solution to system of interest.

NOTE : An enabling system is a system which makes possible the creation, or ongoing availability for use, or the system of interest during some part of its life cycle, e.g. a production system, a maintenance system.

15. Except for simple problems, develop logical solution descriptions (description of how the system is to meet its requirements) as a means of developing physical solution descriptions (description of how to build the system).

16. Be prepared to iterate in design to drive up the benefit, to the applicable stakeholders, of the outcomes of design.

17. Select between (feasible) design alternatives based on the evaluation of expected benefit to applicable stakeholders, i.e., on expected effectiveness.

18. Subject level of risk, independently verify work products (is the job done right, i.e., does the work product meet the requirements for that work product?).

19. Subject to level of risk, independently validate work products (is the right job being done, i.e., does the work product meet the need for that work product?).

20. Some management is needed to plan and implement the effective and efficient transformation of requirements (etc) into solution descriptions - this is unlikely to just happen.

3. Cutler의 원칙 (INCOSE 1997년)

INCOSE 에서는 1997년 8월 5일 로스앤젤레스에서 개최한 INCOSE Symposium의 SE-Principles WG session에서 Cutler는 아래와 같은 8개 항목의 원칙을 제시하였다. 이러한 원칙들은 이미 1995년 7월부터 1997년 1월까지 수 차례에 걸친 워킹그룹내 회의 결과에 따른 것이었다. [5]

1. SE is about building a world. It should be an expression of a world view which is well-founded, complete, and ethical.

2. A system exists by virtue of and to fulfill a purpose for its environment.

3. SE uses mathematical rigor in executing all tasks for which it is appropriate.

4. There is no avoiding process. The only choice is between poor process (freestyling, shortcutting and skipping steps, doing steps out of order) and good process (some realization of a minimal set of representations and manipulations, see below)

5. SE uses formal representations of systems and formal manipulations on those representations to do its job. There is at least one minimal (necessary and sufficient) set of such representations and manipulations.

* Note: SE is itself a system, i.e., self-referential

6. SE, ideally, is a complete and seamless process, i.e., necessary and sufficient, guided by a process framework-roadmap that requires no improvisation, supported by methods and tools which are sufficient for every task and full integrated and compatible.

7. Every system realization has a unique inherent ontogeny. The job of the SE is to discover and facilitate the unfolding of an ontogeny close enough to the inherent ontogeny of a good-enough system.

* Corollary: The ontogeny and end point are interdependent. At each decision point on the outogeny path we choose the branch which seems to lead to the better end point. The image of the end point from any point on the path is a fraction of the path taken to that point.

8. The only product of SE is information (SE does not produce the end system). The information is a set of instructions and supporting data to pre-existing production and operation capabilities that cause the system to be produced, operated, maintained, and improved over its useful life cycle.

4. NCOSE의 원칙 (1993년)

한편 INCOSE의 전신인 NCOSE는 1993년 1월 21일 SE Practice WG 내의 하부 조직인 Subgroup on Pragmatic Principles의 최종 연구결과보고를 통하여 8개 항목에 각각 5 - 12 개 세부항목을 가진 총 72개 항목의 아래와 같은 시스템엔지니어링 원칙을 제시하였다.[6]

여기에서는 지면의 제약을 고려하여 세부항목은 열거하지 않겠다. 세부항목에 관심이 있는 분은 인터넷 주소로 주어진 원전 출처를 방문하여 확인하기 바란다.

"PRAGMATIC PRINCIPLES" OF SYSTEMS ENGINEERING

1. KNOW THE PROBLEM, THE CUSTOMER, AND THE CONSUMER
2. USE EFFECTIVENESS CRITERIA BASED ON NEEDS TO MAKE SYSTEM DECISIONS
3. ESTABLISH AND MANAGE REQUIREMENTS
4. IDENTIFY AND ASSESS ALTERNATIVES SO AS TO CONVERGE ON A SOLUTION
5. VERIFY AND VALIDATE REQUIREMENTS AND SOLUTION PERFORMANCE
6. MAINTAIN THE INTEGRITY OF THE SYSTEM.
7. USE AN ARTICULATED AND DOCUMENTED PROCESS
8. MANAGE AGAINST A PLAN

5. 영국 전기공학회 (IEE) 원칙 (2000년)

한편 영국의 전기공학회인 IEE에서는 1999년에서 2000년 사이에 7회에 걸친 각종 토론회, 세미나, 기타 회의에서 합의한 시스템엔지니어링 원칙들을 12개 항으로 정리하여 제시하고 있는데 그 결과를 보면 아래와 같다.[7] 여기서도 역시 지면의 제약을 고려하여 각 원칙에 대한 부가적인 설명 부분은 생략하겠다.

1. System Purpose

At the initiation of any Systems Engineering programme, a succinct statement of the purpose of the system should be documented and accepted by all stakeholders.

2. System Definition Repository

A repository of information on the system being developed should be managed, controlled and accessible to all parties involved.

3. Parallel Process and Product Development

The Systems Engineering Process should be developed and tailored concurrently with the development of the system products themselves.

4. Multiple Views

Stakeholders will have different perspectives of the system being developed and these viewpoints should be developed as representations of a unified system definition.

5. Progressive Improvement

Each iteration of the Systems Engineering Process should aim to be at least 80% complete.

6. Top Tens

The justification for a project should be characterised in at most ten reasons. The design solution should be justified in at most ten reasons. These reasons may be qualitative, but should be quantified wherever possible.

7. Measure Early

The key to a good Systems Engineering metric is to first decide what needs to be measured and then develop a tool or method to measure it.

8. Synergy with Project Management

Large complex projects benefit significantly from the constructive tension between the Project Manager and Systems Engineer who must communicate well and form a partnership.

9. Human System Integration

Humans should be viewed as part of the system. Their roles should not be defined by default or assumed, but must be evaluated and assessed.

10. Selling Systems Engineering

On a project Systems Engineers must have and use soft skills to sell the Systems Engineering process and product.

11. Emergent Behaviour

The behaviour that emerges when sub-systems are integrated into the super-system should be engineered to maximise desirable and minimise detrimental performance.

12. Architecture Based Work Breakdown

The project Work Breakdown Structure (WBS) should match the target system architecture.

6. Adamsen II의 원칙 (2002년)

아담센 II는 그의 최근 저술인 시스템엔지니어링 교과서에서 한 개 장을 시스템엔지니어링 원칙에 할애하였다. "A Framework for Complex Systems Development" chapter 7. A Potpouri of SDF-Derived Principles를 옮겨 보자. [8]

1. General

1.1 A system architect is responsible to define the interfaces to his/her system, but he/she may not have control over those interface external to his/her system.

1.2 A sound architecture and a successfully managed program consider the total context within which the system must function over its full life cycle.

It is not sufficient to develop an architecture for a deployed system that does not consider all contexts in which the system must function: manufacturing, integration and test, deployment, initialization, normal operations, maintenance operations, special modes and states, and disposal activities.

2. Risk

2.1 Acceptable risk is a key criterion in deciding to move from one activity to the next. The challenge is to quantify risk in an accurate and meaningful way.

2.2 Allocation of resources is a key basis by which to

measure and monitor risk.

Risk in a system development activity can result from insufficient margin in technical, cost, and/or schedule allocation.

2.3 Program risk increases exponentially with requirements instability.

Because upper-level requirements drive lower-level designs and requirements, and because the number of system elements increases exponentially down the program hierarchy, a few unstable top-level requirements can affect many lower-level system elements.

2.4 Risk is difficult to manage if it has not been identified.

2.5 In conceptual architecting, the level of detail needed is defined by the confidence level desired or the acceptable risk level that the concept is feasible.

3. Functional Analysis

3.1 A function cannot be decomposed without some reference to implementation. That which enables the decomposition of a function is knowledge and/or assumptions about its implementation.

3.2 A functional decomposition must be unique. Prescribed functionality describes "what" the system must do and, in order to be self-consistent, that functional description must be unique.

3.3 The functional partitioning is not unique. There are many ways to partition functions.

3.4 The functional definition must include both functionality and associated performance in order to be useful in implementation. It is necessary but not sufficient for implementation to define only required functionality. Performance must also be prescribed in order for a function to be implemented in a meaningful way.

3.5 Within the same tier, the "what" (Requirements Development) drives the "how" (Synthesis Activity). Form must follow function. Implementation, by definition, performs a function. It is not rational to try to determine "how" to perform a function that has not been identified.

3.6 With respect to interaction between hierarchical tiers, the "how" above drives the "what" below. This is a very important principle and has implications in areas such as specification development.

When a customer or next-level-up design team defines functionality in a specification several tiers down, the probability of introducing problems increases because the intermediate decompositions may not be consistent with the prescribed requirements.

4. Allocation

4.1 Margin unknown is margin lost. In order to optimally manage a system development, all system margin must be known to the architect having authority to allocate it. It is generally cost-effective to reallocate resources to handle issues. In order to do this effectively, the architect needs to know where the margin resides.

4.2 During an architecting effort, cost and schedule should be allocated and managed just as any other technical resources.

5. Process

5.1 Process understanding is no substitute for technical understanding. This is exemplified by the principle that a function can not be decomposed apart from implementation. It is the technical understanding of the architecture implementation that facilitates the decomposition.

5.2 Before a process can be improved it must be described.

5.3 Tools should support the system development process, not drive it.

5.4 A central purpose of the SDF is to provide every stakeholder with a pathway for understanding the system and the state of its development.

5.5 There is a necessary order in which technical activities must be performed. Some notion of "what" the system must do must precede any effort to determine "how" to do it.

Some notion of "how" the system will be implemented must precede any determination of system "how well" it performs.

Because trade analyses are based upon cost, schedule, and/or technical criteria, they must follow the synthesis activity.

A trade analysis can not be performed without some definition of implementation.

Trade analyses are based upon cost, schedule, and technical criteria. These can not be determined without some relation to implementation.

5.6 Any technical activity can be categorized as either a "what", "how", "how well", "verify", or "select" activity.

This is the primary organizing principle of the generalized SDF.

5.7 Describing the System Engineering Process in both the time domain (output evolution) and the logical domain (energy distribution) facilitates its application to many contents.

5.8 A system is "any entity within prescribed boundaries that performs work on an input in order to generate an output."

5.9 Given the above definition of "system", the same system development process can be applied at any level of design development.

5.10 The development process must include not only the deployed system but also all necessary supporting entities.

A sound architecture involves the architecture of supporting system elements as well as the element that actually performs the mission functions directly.

Interface can (and should be) defined by the technical process, These help determine roles and responsibilities of teams, information flow and control, subcontract boundaries, etc.

6. Iteration

6.1 Iteration generally occurs for one of three reasons : optimization, derivation, or correction.

Optimization of a design (at some level of fidelity) by definition, results in a change to that design. Where change is necessitated, feed back and/or iteration occurs. This, of cause, is distinct from optimization techniques that are used to develop a design based upon known parameters (e.g., linear, non-linear, and integer programming techniques). Therefore, when optimization is necessitated there is often feed back to the Requirement Development and/or Synthesis activities.

Derivation refer to those situations where lower level design must be done in order to provide the needed confidence level of the architecture at the level above.

Correction covers that broad category of issues that arises as a result of errors.

6.2 Always try re-allocation before redesign.

It is usually less expensive to reallocate resources than it is to redesign.

6.3 The cost of rework increases exponentially with time.

It is relatively easy to modify a specification. It is more difficult to modify a design. It is still more difficult to modify several levels of design and decomposition. It is yet more difficult to modify hardware when it is in manufacturing, still harder during integration and test, still hard once deployed, and so on.

7. Reviews

7.1 A central purpose of a Major Milestone Review is to stabilize the design at the commensurate level of the system hierarchy.

At the first major milestone review, for example, the primary objective ought to be to stabilize the system level architecture and the lower level requirements derived from it. This facilitates proceeding to the next level of design with an acceptable level of risk.

8. Metrics

8.1 A metric's "coefficient of elusivity" is proportional to the definition resolution of the process it is supposed to measure.

The more accurately a process is defined and implemented, the more easily it can be accurately measured.

This is a significant contributor to the difficulty of defining useful system engineering metrics.

Lack of a sufficiently detailed and universal process makes universally applicable metrics difficult to define.

9. Twenty "C's" to Consider.

9.1 CONTROL

Who's minding the store and how?

9.2 CONTEXT

How does this system fit into the next larger system?

9.3 COMMONALITY

Can it be the same for all?

9.4 CONSENSUS

Does the customer agree with our interpretation?

9.5 CREATIVITY

Have all the creative solutions been considered?

9.6 COMPROMISE

Are all system parameters properly balanced?

9.7 CHANGE CONTROL

Are all system impacts understood?

9.8 CONFIGURATION MANAGEMENT

Is everyone working to the current configuration?

9.9 COMPREHENSION

Is the system understood in terms of what it must do and how it works?

9.10 CHARACTERIZATION

Has the required system functionality, performance, and rationale been defined and communicated?

9.11 COHERENCE

Does the system function as an integral whole?

9.12 CONSISTENCY

Have all conflicts been resolved?

9.13 COMPLETENESS

Has the system been fully defined?

9.14 CLARITY

Have all ambiguities been removed?

9.15 COMMUNICATION

Is there good communication between all stakeholders?

9.16 CONTINUITY

Will successors know it was done this way?

9.17 COST EFFECTIVENESS

Is the system over designed?

9.18 COMPETITIVENESS

Can it be done better, faster, cheaper?

9.19 COMPLIANCE

Does the system do what it is required to do?

9.20 CONSCIENCE

Have we done our best?

10. Suggestion for Implementation In Industry

("process du jour syndrome" : reluctance to support yet another three-lettered process to fix all program problems.)

10.1 Implementation of the "SDF" is "tailored" to the specific application by identifying up front what the required input and output will be for each SDF activity.

This can be accomplished with the work sheet, provided in Appendix A, which provides a framework for identifying outputs as well as release status and risk assessment.

10.2 Exit criteria is developed for each Major Milestone Review. These criteria are derived directly from the outputs identified in Chapter 5.

The fidelity or "completeness" of each output is defined as a function of time along the program time line.

In addition, the purpose for each major review is clearly defined, as discussed in Chapter 6.

10.3 Appendix C. Provides an example of Exit Criteria for typical Major Milestone review derived from the outputs of the SDF.

For each Major Milestone Review, the program must produce the generalized output defined in Chapter 5. In so doing, the structured approach is followed by default.

10.4 There are several reasons for moving the implementation strategy in this direction.

While SDF training courses have been, in general, very well received by engineers in industry, there is still a significant element of the "Don't tell me how to do my job" attitude. This is despite the fact that even a cursory evaluation of Chapter 5. must acknowledge that it is only a framework that is constructed.

A second reason for this approach is that the outputs identified are a necessary element of most any development program.

How the outputs are generated is not prescribed, nor is the format in which they must be presented dictated.

It is simply required that they be completed and presented at the Major Milestone Reviews.

7. Rechtin의 원칙 (2002년)

Meir 와 Rechtin은 시스템 아키텍팅 교과서인 "The Art of Systems Architecting"의 부록 A "Heuristics for systems-level architecting"에서 100개가 넘는 시스템엔지니어링 원칙들을 열거하였다.[9]

여기에서는 지면의 제약을 고려하여 일일이 열거하는 것은 생략하기로 한다.

대체로 그 스타일을 짐작할 수 있도록 몇 개만을 소개한다면 아래와 같다.

1. Multitask heuristics

1.1 (D) Performance, cost, and schedule can not be specified independently.

At least one of the three must depend on others.

1.2 (D) With few exceptions, schedule delays will be accepted grudgingly; cost overruns will not, and for good reason.

1.3 (D) The time to completion is proportional to the ratio of the time spent to the time planned to date.

The grater the ratio, the longer the time to go.

1.4 (D) Relationships among the elements are what give systems their added value.

1.5 (D) Efficiency is inversely proportional to universality.

1.6 (D) Murphy's law, "If anything can go wrong, it will."

1.6.1 (P) Simplify, Simplify, Simplify.

1.6.2 (P) The first line of defense against complexity is simplicity of design.

1.6.3 (P) Simplify, combine, and eliminate.

1.6.4 (P) Simplify with smarter elements.

1.6.5 (P) The most reliable parton on an airplane is the one that isn't there - because it isn't needed.

1.7 (D) One person's architecture is another person's detail. One person's system is another person's component.

1.7.1 (P) In order to understand anything, you must not try to understand everything.

1.8 (P) Don't confuse the functioning of the parts for the functioning of the system.

1.9 (D) In general, each system level provides a context for the levels below.

1.9.1 (P) Leave the specialties to the specialists. The level of detail required by the architect is only to the depth of an element or component critical to the system.

But the architect must have access to that level and know, or be informed, about its criticality and status.

1.9.2 (P) Complex systems will develop and evolve within an overall architecture much more rapidly if there are stable intermediate forms than if there are not.

1.10 (D) Particularly for social systems, it is the perceptions, not the facts, that count.

1.11 (D) In introducing technological and social change, how you do it is often more important than what you do.

1.11.1 (P) If social cooperation is required, the way in which a system is implemented and introduced must be an integral part of architecture.

1.12 (D) If the politics don't fly, the hardware never will.

1.12.1 (D) Politics, not the technology, set the limits of what technology is allowed to achieve.

1.12.2 (D) Cost rules.

1.12.3 (D) A strong, coherent constituency is essential.

1.12.4 (D) Technical problems become political problems.

1.12.5 (D) There is no such thing as a purely technical problem.

1.12. 6 (D) The best engineering solutions are not necessarily the best political solutions.

1.13 (D) Good products are not enough. Implementations matter.

1.13.1 (P) To remain competitive, determine and control the keys to the architecture from very beginning.

2. Scoping and Planning

2.1 The beginning is the most important part of the work.

2.2 Scope! Scope! Scope!

8. 원칙들의 비교

이상 위에서 열거한 바와 같이 시스템엔지니어링 원칙들은 다양한 모양으로 제시되고 있다. 어떤 사람은 10 개도 안 되는 소수의 원칙을 제시하는가 하면 다른 사람은 100 개가 넘는 수많은 원칙들을 열거한다. 그러나 대부분은 현장 경험에서 교훈을 도출한 실용적 원칙들이지 학문적인 연구의 결과로 얻어진 철학적 원칙 또는 수학적 원칙들은 아니다.

여기서 실용적 원칙에는 다시 현장의 시스템엔지니어링에 관한 실태를 사실대로 묘사하는 서술적 원칙들이 있는가 하면, 곧바로 흔히 일어나는 실수를 회피하거나 난관을 극복하는데 사용할 수 있는 처방적 원칙들도 있다.

철학적 원칙이란 어떤 것일까?

아직 시스템엔지니어링의 진수를 알지 못하는 입장에서 시스템엔지니어링의 철학적 원칙 또는 원리가 이것이다 라고 적절한 예를 들지 못함이 안타깝다. 그러나 본 저자에게 보다 익숙한 물리학의 예를 든다면 그나마도 이해가 가능할지도 모르겠다.

물리학에서 철학적인 기분을 느끼게 하는 문제가 야기된 것은 20세기 시작되면서 나타난 두 가지 문제가 시작하면서였다. 그 하나는 양자론이요 다른 하나는 상대론이다. 이 두 문제는 원래부터 인간의 감각적인 범위를 넘어서는 것을 대상으로 하고 있었기 때문에, 또한 우주의 탄생과 종말, 우주의 중심과 주변이라는 주제와 깊게 연관되어 있었기 때문에 처음부터 철학적인 냄새가 짙게 배어 있었는지 모른다. 그러나, 더욱 철학적인 냄새를 본격적으로 피우기 시작한 것은 양자론에 있어서 "불확정성의 원리"라는 것이 제시되면서다. 그러나 물리학에서 가장 철학적인 원리는 바로 보어의 "상보성 원리" (complementary principle)라는 것으로 생각된다.

간단히 말하자면, 그것이 상대론이든 양자론이든 어떤 물리학적 이론이 새롭게 인정받기 위해서는, 기존의 물리학에서 여태까지 인정받아온 과거의 이론, 예를 들면 뉴턴의 중력에 관한 법칙이나 힘과 운동에 관한 법칙 등과 상충되어서는 안 된다는 것이다.

예를 들어서 상대성 이론이라는 것은 빛 알맹이의 운동에 대해 기존의 물리학이 설명할 수 없었던 부분을 명쾌하게 설명하면서 동시에 에너지와 질량이 결국 같은 것이라는 등 새로운 개념을 만들어내는데 공을 세웠다. 그러나 한편 상대론은 기존에 다루었던 "일상적인 범위"에 대해서는 뉴턴의 중력의 법칙이나 운동 법칙과 동일한 결과를 나타냄으로써 기존의 물리학적 성과를 인정하고 포용하는 결과가 되었다. 물론, 기존의 물리학이 다루지 못했던 "빛의 속도에 가까운 특별한 범위"에 대해서는 보다 더 정확한, 사실에 부합하는 결과를 나타냈기에 기존의 이론들보다는 더 훌륭한 발전된 이론이라는 것이다.

이렇게 되었을 때에 상대론과 같은 새로운 이론은 기존의 물리학의 성과를 인정하고 포용하는 바탕 위에서 보다 발전적인 새로운 이론으로 인정될 수 있다는 것이다.

이러한 "상보성 원리"라는 것은 어떤 물리현상을 직접 설명하는 원칙은 아니지만, 어떤 중요한 원칙보다도 더 중요한, 철학적인 성격을 갖는 근본 원칙이라고 볼 수 있을 것이다.

과연 시스템엔지니어링에서는 어떤 원칙이 이러한 철학적 원칙으로 인정받을 근본적인 문제를 다룬다고 볼 수 있는가? 아마도 물리학의 역사가 수 천년 또는 수 백년이고, 현대 물리학의 역사만 하여도 100 년이 넘는 것을 감안하고 또 시스템엔지니어링의 역사가 겨우 수십년이고 INCOSE의 역사가 10여년 정도인 점을 인정한다면 시스템엔지니어링 분야에서 아직 철학적인 원리, 원칙을 찾아내기 어렵다고 하여 너무 실망할 필요는 없을 것으로 보인다.

INCOSE 2002 라스베가스 심포지엄에서 있었던 Systems Engineering Fundamentals 강좌에서 Eliot Axelband라는 USC 교수는 "시스템엔지니어링은 왜 어려운가?"에 대하여 이렇게 설명하였다. [10]

1. 시스템엔지니어링에서 다루는 시스템은 복잡하고 혁신적인 시스템이다.
2. 시스템의 아키텍처는 임의적이다. (정답이 없다)
3. 인터페이스가 중요한데 처음에는 파악이 안 된다.
4. 전문분야 기술에도 능통해야하고 시스템엔지니어링 프로세스에도 능숙해야 한다.
5. 시스템엔지니어링은 시스템의 전체 수명주기를 다루어야 한다.
6. 시스템엔지니어링은 "욕망에서 먼지로의 가혹한 행군"이며, 비 해석학적 톱-다운 설계과정이다.
7. 해결책의 개념이 유일하지 않다. (정답이 없다)
8. 최적화는 불가능하며, 다만 판정에서 합격을 바랄 뿐이다.
9. 사회적(정치적) 요소가 중요한데 예측하기 어렵다.
10. 형태와 기능 사이에서 실험적인 반복이 필요하다.
11. "재 설계 역시 설계의 한 부분이다"
12. 상향식 통합 및 시험 과정
13. 방대한 자료, 형상관리의 필요성, 특히 진화적 획득

의 복잡성

14. 시스템 아키텍팅이나 엔지니어링은 과학을 이용하는 예술이다.

이처럼 시스템엔지니어링이란 원래 복잡한 시스템, 그것도 여태까지 경험해보지 못한 전혀 새로운 시스템을 다루기 때문에 어려울 수밖에 없는 것이 당연하다. 오히려 쉽다면 그것은 시스템엔지니어링이 아니라는 말이나 마찬가지이다.

문제는 이처럼 원래부터 복잡한 과제를 가능한 쉽게 풀어나가는데 시스템엔지니어링의 존재가치가 있고, 그것이 바로 시스템엔지니어링의 강점인 반면에, 복잡한 것을 쉽게 풀어나가는 노하우, 또는 문제에 대한 해답이나 그 접근방법도 유일한 것이 아니라는 애매한 상황에서 더욱 복잡성이 증가한다.

따라서 현장의 실패 및 성공에서 얻은 값비싼 경험 요소와, 이러한 경험요소를 간결한 언어로 표현한 시스템엔지니어링의 실용적 원칙들이야말로 시스템엔지니어들에게는 더할 수없이 귀한 보배라 할 것이다.

이를 더욱 더 귀하게 여기고 수집하고 보석처럼 갈고 닦아 더욱 빛나는 보배로 만들 수 있다면 머지않아 우리 시스템엔지니어링에도 철학적 원리가 나타나리라고 기대해 본다.

9. 결론

이상 현재까지 여러 문헌에서 수집한 시스템엔지니어링 원칙들을 비교 검토한 결과에 의하면 아직까지 INCOSE에서 한 때 시도한 바 있는 수학적 원칙, 철학적 원칙들은 발견되지 않고 있으며, 오직 실용적 원칙들만 존재한다는 것을 알게 된다.

이들 실용적인 원칙들은 현장의 경험을 요약한 것이기 때문에 많으면 많을수록 도움이 된다고도 볼 수 있을 것이다. 그런 의미에서는 Rechtin 등이 제시한 100 가지가 넘는 원칙도 활용하는 사람에 따라서는 큰 도움이 되는 원칙이 될 수 있다.

그러나 시스템엔지니어링 전반의 교훈을 담은 원칙이라면 10 가지 내외로 정리하는 것이 바람직하다. 이것이 바로 INCOSE, IEE, 또는 Halligan 등이 제시한 시스템엔지니어링 원칙들이라고 볼 수 있다.

시스템엔지니어링을 실제로 수행하는 현장 전문가들은 현장의 경험을 통하여 얻는 귀중한 교훈들을 압축하여 새로운 실용적 원칙으로 제시하고, 시스템엔지니어링을 연구하는 학자들은 이들 다양한 원칙들을 비교 검토하여 10개 내외의 통합된 원칙으로 압축하여 보다 철학적인 시스템엔지니어링 정신을 담아낼 수 있도록 하는 것이 바람직하다는 생각이다.

참고문헌

1) Halligan, Robert., "Systems Engineering Principles - Essence and Implementation", 2002 시스템엔지니어링 workshop 논문집, 한국시스템엔지니어링협회 (2002.11)

2) INCOSE SE-principles working group Meeting. http://www. incose. org/ workgrps/ pwg/ Meetings/ LA_Meeting. html

3) Meir, Mark W. and Rechtin, Eberhardt, "The Art of Systems Architecting" 2nd ed. 2002 CRC , Appendix A "Heuristics for systems-level architecting"

4) Halligan, Robert., "Systems Engineering Principles - Essence and Implementation", 2002 시스템엔지니어링 workshop, 한국시스템엔지니어링협회 (2002. 11)

5) Cutler, William., Presented at INCOSE Principles WG Session, Tuesday, August, 5th, 1997, Los Angeles, CA, http://www.incose.org/ workgrps/ pwg/ Meetings/ Meeting.html

6) DeFoe, J.C., editor, "An Identification of Pragmatic Principles - Final Report", INCOSE : SE Principles Working Group' Subgroup on Pragmatic Principles (1993.1.21), http://www. incose. org/ workgrps/ practice/ pragprin. html

7) The Institution of Electrical Engineers : Professional Network : PN Sector - Management : Systems Engineering , http:// www. iee. org/ onComms/ pn/ systemeng/ principles. cfm

8) Adamsen II, Paul B., "A Framework for Complex System Development", chapter 7, CRC press (2000)

9) Mark W. Meir and Eberhardt Rechtin, "The Art of Systems Architecting" 2nd ed. 2002 CRC , Appendix A "Heuristics for systems-level architecting"

10) Axelband, 2002 INCOSE Symposium procedings.