Course: **Introduction to Engineering**

Unit: **System Engineering**

Author: **Dr. Johan Gouws**

B.Eng. & M.Eng. (Elec.) (Rand Afrikaans University, South Africa)

MBA (Heriot-Watt University, Scotland)

Ph.D. (Wageningen, the Netherlands)

Issue number: **1.1**

Date: **March 2007**

TABLE OF CONTENTS

1. INTRODUCTION

1.1 IMPORTANCE OF SYSTEM ENGINEERING

- *System level thinking* was listed in Lecture 1 as one of the essential skills of a professional engineer.
- In order to manage large scale system development, the concept *system engineering* had been developed – initially for use in the defence, aviation, and space industries.
- Even for those engineers who won't work on such large scale systems, the ability to think about "the bigger picture" is an invaluable skill – and that is what system engineering will help you do.

1.2 SCOPE OF THIS LECTURE

This lecture introduces the following concepts:

- The system engineering process.
- The system life-cycle (from requirements definition to eventual phase-out).
- System hierarchies (breaking down complex systems into more manageable chunks).
- The importance of requirements definition before system development can commence.
- System development models (from initial prototype to production model).
- Specifications for different system development stages.

2. THE SYSTEM ENGINEERING PROCESS

- System engineering is an extensive and complex process, best applied to large development projects such as those undertaken in the military-, space- and aviation industries.
- The *system engineering process* can be defined as the application of scientific and engineering efforts to:
  - Transform a user requirement into a system specification - which defines system performance parameters and the preferred system components and system configuration.
  - Co-ordinate the acquisition of the defined system components, and their integration into a system which satisfies the user requirement.
- System engineering is a multi-faceted, **iterative** process of problem definition, analysis, design, construction, test and evaluation, commissioning, operational support, and finally phase-out and disposal.
- It requires attention to a wide variety of aspects such as interfacing, reliability, availability, maintainability, safety, survivability, producibility, ergonomics, etc.
- To co-ordinate the system engineering process, a *System Engineer* - supported by a team of technical specialists - is required.
- The System Engineer always works closely with a Project Manager.
- The System Engineer's responsibilities (supported by a technical team) typically include:

- Requirements definition and –analysis (including a feasibility study).
- Design (preliminary / concept design, and detailed design).
- Initial implementation (build, integrate, test)
- Support during production.
- Support during delivery and commissioning.
- Support during maintenance / enhancement.
- Support during phase-out and disposal.

- The Project Manager's responsibilities typically include:
  - Project planning (task breakdown, budgeting, and scheduling).
  - Organizing (choosing a suitable organizational structure).
  - Staffing (allocation of responsibilities, and filling the organizational positions).
  - Leading / Directing the team members.
  - Controlling the work and the progress.
- After initial development, a Product Manager also joins the management team, to take charge of final development, production, delivery and commissioning, maintenance, and eventual phase-out.
- The system engineering process can only be executed sensibly if it is done systematically according to a well defined *System Engineering Management Plan (SEMP)* – see GOUWS & GOUWS 2006 for an example of a framework for such a plan.
- Execution of the plan normally takes place in phases, and will result in various other system engineering reports – all aimed at producing a solution to satisfy the user requirement, within the constraints of cost, schedule, and achievable technical performance.
- More information about system engineering can be found in sources such as ASLAKSEN & BELCHER 1992, BLANCHARD & FABRYCKY 1990, BOARDMAN 1990, KASSER 1995, KAYTON 1997, RHOADS 1983, and VERMA & FABRYCKY 1997.

3. SYSTEM LIFE CYCLE

- The development and utilisation of any complex engineering system typically take place in four main life cycle phases, as shown in Table 1 and Table 2.
- Names of reports for which frameworks are provided by GOUWS & GOUWS 2006 are printed in ***bold italics*** in Table 2.
- Overlaps between the life cycle phases, and iteration, are often essential.
- After each major system engineering phase, all reports compiled up to that stage should be thoroughly reviewed and updated where necessary.

Table 1: Four main phases in the typical system life cycle

REQUIREMENTS FORMULATION	ACQUISITION	UTILISATION AND SUPPORT	PHASE-OUT AND DISPOSAL

Table 2: Expansion of system life cycle phases

Life Cycle Phase	Typical Activities	Typical Outputs
REQUIREMENTS FORMULATION	• Obtain a need statement (*wish list*) from the end user / client • Analyse the current situation (available solutions, state-of-the-art, etc.) • Define the desired situation (ideal solution for the client's needs) • Compare the desired situation with the current situation, and define the gaps	Gap analysis report
	Formulate the problem for which a solution must be found by means of the system engineering process	***User Requirements Statement (URS)***
ACQUISITION (1) Concept exploration (Conceptual design)	Translate the user requirements into system-level functional requirements	***System Specification***
	Compile a plan for systematically managing the system development	***System Engineering Management Plan (SEMP)***
	Identify and compare alternative concepts which could satisfy the functional requirements	***Trade-off study report***
	Design suitable laboratory experiments in order to evaluate different concepts	Engineering design notes (***Technical Notes***)
	Design, build and test chosen concepts (with emphasis on functionality) – often called prototyping	• ***Design reports*** • Exploratory development model (XDM) • ***Test and Evaluation Report***

Life Cycle Phase	Typical Activities	Typical Outputs
ACQUISITION (2) Demonstration and validation (Preliminary design / Advanced development)	Design and specify subsystems to perform the desired system functions (as described in the system specification)	• Final ***System Specification*** • ***Development Specifications***
	Upgrade design of XDM subsystems in order to better perform the system functions	***Design reports***
	Specify interfaces between subsystems	***Interface Control Document (ICD)***
	Define overall test and evaluation requirements for the system	***Test and Evaluation Master Plan (TEMP)***
	Define functional test requirements	• Functional test specification • Functional test procedure
	Develop different subsystems and interface them according to interface control report	Advanced development model (ADM)
	Evaluate advanced development model against functional test requirements and report the test results	• Functional test log • ***Test and Evaluation Report***
ACQUISITION (3) Full-scale engineering development (Detail design and development)	Specify physical requirements (*'form"* and *"fit"*)	• Final ***Development Specification*** • ***Product Specifications*** • Process and Material Specifications
	Upgrade design of ADM in order to better perform the system functions, and to meet *form* and *fit* requirements too	***Design reports***
	Specify interfaces between the system and its intended environment and extend the interface control report	Updated ***Interface Control Document (ICD)***
	Analyse and define appropriate logistic support concepts	***Integrated Logistic Support Plan (ILSP)***
	Refine functional test requirements, and define operational test requirements	• Acceptance test specification • Acceptance test procedure

Life Cycle Phase	Typical Activities	Typical Outputs
	Redevelop subsystems and interfaces (where necessary) to ensure *form, fit* and *function* (F^3) compliance	Engineering development model (EDM)
	Evaluate EDM against acceptance test requirements and report test results; and update the TEMP	• Acceptance test log • Updated ***TEMP*** • ***Test and Evaluation Report***
ACQUISITION (4) Industrialisation	Compile a production data pack for the designed (*created*) system; and define a manufacturing plan	• Final Product-, Process- and Material Specifications • Engineering drawings; parts lists; assembly- and commissioning instructions; etc. • Manufacturing plan
	Define and design the production (*creating*) system. (This can be a separate process, similar to the above requirements formulation and system acquisition steps.)	Process flow diagrams; production facility definition; production instructions; etc.
	Upgrade design of EDM (where necessary)	***Design reports***
	Finalise system design, and build a prototype of future production models	Pre-production model (PPM)
	Compile operating manuals	User manuals; maintenance procedures; training manuals; etc.
ACQUISITION (5) Production	Manufacture the required number of systems according to the manufacturing plan	Production models (PMs)
	Perform system configuration management by keeping an accurate *"as-built record"* for each system	Build-history for each system
	Do engineering analyses to substantiate any changes to system configuration	***Design reports***
ACQUISITION (6) Commissioning	Compile a commissioning plan and commissioning procedures	• Commissioning plan • Commissioning instructions
	Commission the production models, according to the commissioning plan and -instructions	Commissioned systems

Life Cycle Phase	Typical Activities	Typical Outputs
	Evaluate a selected number of production models against acceptance test requirements and report test results	• Acceptance test logs • ***Test and Evaluation Report***
UTILISATION AND SUPPORT	Utilise the system in its intended operational role	Operational results
	Perform logistic support analysis for a system already in operation (if necessary)	Logistic Support Analysis Records
	Gather operational data for the management information system; and use it to: • ensure that the system satisfies its requirements; and • initiate design changes as necessary	• Operational data in the management information system • Engineering change proposals
	Plan and implement maintenance actions as necessary	• Maintenance management plan • Maintenance data • Maintenance records
PHASE-OUT AND DISPOSAL	Evaluate factors such as: • continued existence of operational requirements • matching between operational requirements and system performance • feasibility of phase-out or not (including cost of keeping system in operation versus replacement cost) • availability of alternative systems	Engineering and management evaluation reports
	Compile a phase-out and disposal plan (including an environmental impact study - if necessary)	Phase-out and disposal plan
	Support phase-out and disposal of the system	Phased-out and disposed systems

4. SYSTEM HIERARCHIES

- Before system development can begin, it is necessary to use a top-down approach to organise the system into successive layers of functional units, and to assign responsibility for development of the different units to different suppliers.
- This is a form of breaking down the problem into smaller problems (analysis).
- The term *system* is often used generically, which can be very confusing, since one person's *system* can be several layers down in another's system hierarchy.
- The only way to alleviate this problem is to always ensure that the "system" is so well described

that there can be no doubt where its boundaries are, and where it fits into the *bigger picture*.

- RHOADS 1983 defined a top-down system hierarchy as in Table 3.
- Downwards from the fourth level of this hierarchy, a distinction is made between hardware and software.
- Sometimes a further hardware level is added at the bottom, namely *characteristics, materials and processes.*
- These definitions are not standardised at all, and especially with rapid developments in software engineering, different software terminologies come into use continuously.
- The generic term *item* is often used to refer to a system, or a segment, or an element, etc.
- The term *configuration item* is normally used to indicate an item that is custom-made for a specific system (and to which a serial number is allocated).
- Off-the-shelf components are normally not configuration items.
- In some systems, there are configuration items right down to the lowest level of the system hierarchy - e.g. a specially designed integrated circuit.
- In other systems, the configuration items are limited to the upper levels of the system hierarchy - e.g. only down to engine level in many vehicles.
- An alternative, bottom-up definition of a system hierarchy is shown in Table 4.

Table 3: Definition of a typical top-down system hierarchy

LEVEL	***DESCRIPTION***	
1	***System***: Hardware, software, material, facilities, personnel, data and services needed to perform a designated function with specified results - e.g. an automated dairying system, consisting of milking robots, automated animal feeders, data loggers, support personnel, cows, etc.	
2	***Segment***: A grouping of elements which are closely related and normally physically interfaced. These elements are typically produced by several contractors and then integrated by one contractor - e.g. a milking robot, consisting of a machine vision element and a robotic manipulator element.	
3	***Element***: A complete, integrated set of subsystems, delivered by a single contractor, and capable of accomplishing a specific function - e.g. the manipulator for a milking robot, consisting of a robot arm and robot hand.	
4	***Subsystem***: A functional grouping of components to perform a major function within an element - e.g. a robot hand, consisting of a gripper and actuators.	***Program***: A major, independent part of software to perform a major function in an element - e.g. a robot manipulator control program.

LEVEL	*DESCRIPTION*	
5	***Component***: A functional unit viewed as an entity for purposes of analysis, manufacturing, testing, maintenance, or record keeping - e.g. an actuator for a robot hand.	***Subprogram***: A major functional subset of a program - e.g. robot hand control software.
6	***Subassembly***: Two or more units joined together to form a stockable unit suitable for disassembly or part replacement - e.g. a printed circuit board with parts mounted on it.	***Module***: An independently compilable software component, made up of procedures or routines.
7	***Part***: A single piece not normally subject to disassembly without destruction - e.g. resistors, transistors, nuts and bolts, etc.	***Procedure***: A standardised sequence of statements to accomplish a specific function; or ***Routine***: A standardised sequence of statements to accomplish a repeatedly used function.

Table 4: Definition of a bottom-up system hierarchy

LEVEL	*DESCRIPTION*
1	Characteristics / Materials / Processes
2	Components (e.g. electronic components)
3	Product subsystems (e.g. a control card for an electrical power supply)
4	Products (e.g. a complete electrical power supply)
5	Product systems (e.g. a process controller consisting of electrical power supplies, computer, measurement sensors, etc.)
6	User system (e.g. the combination of power stations, a power distribution network, and support personnel)
7	User system group (more than one user system interfacing with each other)
8	Global systems (more than one user system group interfacing with each other)

5. REQUIREMENTS DEFINITION

5.1 USER REQUIREMENTS STATEMENT

- A *user requirements statement* (URS) is a description of operational needs experienced by a specific user; and signals the start of the system engineering process.
- A very simple example is: *"Transport 10 people to work every morning, over a distance of 40*

km, within one hour; and back home again in the afternoons".

- Depending on the user's technical abilities, the statement can range from a vague *wish list* (as above), to a detailed list of operational requirements.
- In general only the user's needs should be stated and not any specific solution - unless the user specifically wants to include or exclude certain potential solutions.
- It is the purpose of the system engineering process to translate the requirements to real technical terms, to identify potential solutions, and to find the best solution for the stated requirements.
- The URS structures the acquisition process, and should be formulated very carefully.
- If the wrong problem is defined at the beginning, the whole acquisition process is jeopardised.
- For software development, a *software requirements specification* (SRS) is compiled.
- The basic principles used to compile an SRS are the same as those used for compiling a URS.

5.2 GOAL OF THE REQUIREMENTS DEFINITION PHASE

- The goal of a project's requirements definition phase is to decide **what to do**, and to document the results of the decision-making process.
- This step starts with listening to what the client wants – remember *people buy what they want, not what you think they need*.
- Often the client's wish list is very elaborate, or the client is unsure about their requirements.
- Requirements definition must always be an iterative process, where the client is guided in order to remain realistic in terms of requirements, cost and schedule.
- Defining requirements is one of the **most difficult** parts of any project; and also the part with the most **disastrous consequences when done poorly**.
- Yet, requirements definition is **often neglected**, since design engineers are eager to "get going".
- Requirements definition is not enough. Requirements must also be analysed carefully to ensure that they are realistic, and that they are realizable.

5.3 PERSPECTIVES ON REQUIREMENTS

- A URS and an SRS are essential documents, which must be compiled before system development can begin.
- An URS / SRS has different functions for different people, e.g.:
  - For **clients**: definition of what to expect / contractual baseline.
  - For **managers**: establishes feasibility of the project, and serves as a basis for planning and controlling the project.
  - For **designers**: "design-to" specification.
  - For **implementers**: defines range of acceptable implementations, and the desired outputs.
  - For **quality assurance** personnel: basis for validation, test planning, and verification.

- Requirements can often be classified as:
  - "**Functional**": mapping between inputs and outputs (i.e. what the system must do).
  - "**Non-functional**": all other constraints, including maintainability, safety, modularity, etc.
- Alternatively requirements can be classified as:
  - "**Behavioural requirements**": defines operational behaviour of the system - e.g. transformation of inputs to outputs, timing, safety, fault tolerance, etc.
  - "**Developmental quality attributes**": defines characteristics resulting from the development process - e.g. maintainability, modularity, testability, etc.

5.4 EFFECTS OF IMPROPER REQUIREMENTS DEFINITION

- Requirements definition is a **critical phase** in system development, as illustrated by the following quotations, from sources spanning many years, about software development:
  - *"In nearly every software project that fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in project failure"* – ALFORD & LAWSON 1979.
  - Studies of problems with mission-critical defence systems in the USA, identified inadequate requirements as a major problem area in 2/3 of the cases – US-GAO 1992.
  - A major source of safety-related software errors in NASA's Voyager and Galileo spacecraft was found to be improper functional and interface requirement definitions – LUTZ 1993.
- For software development, an error costing $1 to fix during the requirements phase typically costs (FAULK 1997):
  - $2 to fix during the concept design phase.
  - $5 to fix during the detailed design phase.
  - $10 to fix during the coding phase.
  - $20 to fix during the unit test phase.
  - $50 to fix during the system test phase.
  - $100 to fix during the deployment phase.
  - $200 to fix during the maintenance phase.
- The fact that requirements definition problems still exist today, in spite of extensive system engineering practices, does not indicate that these practices are useless.
- It indicates that the extent of systems is growing at a similar rate than system engineering practices; and that further development and application of the latter needs constant attention.

5.5 DIFFICULTIES WITH REQUIREMENTS DEFINITION

- *"The hardest single act of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements … No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."* – BROOKS 1987.
- The requirements phase must establish and specify **WHAT** the system must do, without

prescribing HOW to do it.

- However, it can be very difficult to distinguish "what" and "how" from each other.
- One person's "how" can be another's "what".
- "How" on one level of system detail can be "what" on another level.
- Agreement on definition of different system levels therefore becomes very important – but this issue is project-size dependent.
- For large, multi-person, multi-version system development, requirements specification can be divided into sub-goals:
  - Understand precisely what the system must do.
  - Communicate this understanding to all parties involved.
  - Iterate, and document the process.
- There are two main types of difficulties associated with system development – as shown in Table 5.
- It is important to identify the project difficulties early, in order to find the best route to address these difficulties.

Table 5: Type of difficulties associated with system development

Difficulty	Factors Contributing to the Difficulty	Alleviated by
Inherent / internal / essential (inherent to the problem to be solved)	• Comprehension: often clients are unsure of what exactly they want; or system developers find it difficult to comprehend the requirements • Uncertainty about the difference between the client's *wants* and *needs* • Communication: requirements are difficult to communicate effectively, since the system can be very difficult to visualise, and since different people must use the user requirements statement. • Inseparable concerns: It's often difficult to separate issues and deal with them one by one	Experience / skills / expertise
External / accidental (created by inadequate methods and processes)	• Requirements documented superficially or only after system development merely to keep somebody happy. (This happens when developers, clients and managers quickly wanted to see "results" instead of mere "paperwork".) • Having an requirements statement that attempts to be all things to all people • Trying to baseline and freeze requirements too soon, instead of allowing for changes	Structured system engineering process

5.6 PROBLEM ANALYSIS AND RATIONAL DECISION-MAKING

- Very often, problem analysis is not done properly, in order to get going as soon as possible, with a new project.
- Remember: it is better to solve a well-defined problem halfway, than to solve the wrong problem in full.
- In order to avoid wonderful solutions looking for suitable problems, or costly reworking, proper problem analysis and rational decision-making are essential.
- Problem analysis and decision-making are not something that a single manager can do in isolation.
- It should be a team-effort - to remove personal biases, and to reduce the risk of oversights.
- It requires a proper scanning of the internal and external environments in order to:
  - Determine user *wants* and *needs*, and understand these.
  - Determine what inputs are available.
  - Translate user requirements to desired outputs, and ways of achieving these.
  - Decide what activities are required in order to best convert available inputs to desired outputs.
  - Determine opportunities and threats.

5.7 ENSURING A PROPER REQUIREMENTS STATEMENT

- In general a **complete system requirements approach** will define:
  - **Process** (the *creating system*): list of activities and their sequence, entrance and exit criteria, desired results, and resource requirements for each activity.
  - **Product** (the *created system*), with acceptance criteria for the system.
- Generic properties for a **proper requirements statement** include:
  - It must be *complete* (unavailable information must be marked "TBD" [*to be determined*]).
  - *Implementation independent* (as far as possible) - i.e. specify WHAT, and not HOW.
  - *Unambiguous* and *consistent*.
  - *Precise* (define desired behaviour exactly).
  - *Verifiable*.
- Characteristics of a proper SRS (how it is written):
  - **Modifiable** (requirements changes are inevitable).
  - **Readable**.
  - **Organised** for reference and review (engineers and reviewers must be able to find information in the document quickly).
- A proper requirements statement provides:

- A **basis for standardising** an organization's processes and products.
- A **standard for progress measurement**.
- **Guidance to developers** regarding what needs to be done next.

- Although requirements are often not fully understood before implementation starts, and a perfect requirements statement cannot be made, these are not reasons to neglect the process in total.
- The development of technical specifications is an essential part of disciplined engineering.
- Without the requirements in writing, no sensible communication can take place.
- Unless system requirements, and the goals and objectives, can first be written down in common language, it cannot be converted into a system.

6. SYSTEM DEVELOPMENT MODELS

- Table 2 showed various development models that typically result from the system development process.
- These models are summarised in Table 6.

Table 6: Summary of system development models

Life Cycle Phase	Typical Activities	Development Model
ACQUISITION (1) Concept exploration (Conceptual design)	Design, build and test chosen concepts (with emphasis on functionality)	Exploratory development model (XDM)
ACQUISITION (2) Demonstration and validation (Preliminary design / Advanced development)	Develop different subsystems and interface them according to interface control report	Advanced development model (ADM)
ACQUISITION (3) Full-scale engineering development (Detail design and development)	Redevelop subsystems and interfaces (where necessary) to ensure *form, fit* and *function (F^3)* compliance	Engineering development model (EDM)
ACQUISITION (4) Industrialisation	Finalise system design, and build a prototype of future production models	Pre-production model (PPM)
ACQUISITION (5) Production	Manufacture the required number of systems according to the manufacturing plan	Production models (PMs)

7. SPECIFICATIONS

- From Table 2 it is clear that some form of *specification* is often an output of one of the acquisition stages in the system life cycle.
- **Error! Reference source not found.**7 lists different types of specifications, as defined by MIL-STD-490.
- (Although the system engineering process originated in the defence industry, it is just as applicable to other engineering industries – as long as tailoring is used to adapt to specific circumstances.)
- Sometimes a system is too large and too complex to compile a single *system specification* for it, in which case different *segment specifications* can be compiled at one level below the overall system level.
- A segment specification has the same format and similar purpose as a system specification. At lower levels of the system hierarchy, *development* and *product specifications* are typically compiled.
- It is important to remember that a specification is not a novel, but a series of requirement statements, which should stipulate all essential technical requirements, be unambiguous, and be realistic with its requirements.

Table 7: Different types of specifications

Type	Specification	Purpose
A	**System specification**	• States mission and user requirements for the system as an entity, in technical terms. • Also compiled at segment level (refer to Table 3). • Defines functional requirements (*what* the *total system* must do). • Groups functional requirements, allocates these to configuration items, and defines interfaces between these items and with system's environment.
B B1 B2 B3 B4 B5	**Development specification** Prime Item Development Critical Item Development Non-complex Item Development Facility Development Computer Program Development	• One prime item development specification per major component defined in a system specification. • One critical item development specification per major component defined in a prime item development specification. • States design requirements for each configuration item (i.e. *what* the *configuration item* must do) - for use by system designers.

Type	Specification	Purpose
C C1a C1b C2a C2b C3 C4 C5	**Product specification** Prime Item Product Function Prime Item Product Fabrication Critical Item Product Function Critical Item Product Fabrication Non-complex Item Product Fabrication Inventory Item Computer Program Product	• One product specification per major component defined in each development specification. • Specifies physical characteristics: ∗ either in a form suitable for procurement (i.e. what the configuration item must look like and what it must do) - for use by procurement personnel; ∗ or as designed by system designers according to requirements in a development specification (i.e. how the configuration item must be made) - for use by fabrication (production) personnel.
D	**Process specification**	• Defines non-standard processes, which are critical for correct manufacture of an item. (Standard processes should always be used as far as possible.) • Acts as a constraint during industrialisation.
E	**Material specification**	• Defines non-standard materials, which are critical for correct manufacture of an item. (Standard materials should always be used as far as possible.) • Acts as a constraint during industrialisation.

- Guidelines for the preparation of system-, development-, and product specifications are provided by GOUWS & GOUWS 2006.
- Although both hardware and software can be covered by these specification types, the guidelines provided are mainly for hardware development.
- (Also refer to WYSS 1988 for guidelines on specification writing.)

8. REFERENCES

ALFORD, M. & LAWSON, J. (1979): *Software Requirements Engineering Methodology (Development);* RADC-TR-79-168, US Air Force, Rome Air Development Center

ASLAKSEN, E. & BELCHER, R. (1992): *Systems Engineering*; Prentice Hall

BLANCHARD, B.S. & FABRYCKY, W.J. (1990): *Systems Engineering and Analysis* (Second Edition); Prentice Hall

BLANCHARD, B.S. (1986): *Logistics Engineering and Management* (Third Edition); Prentice Hall

BOARDMAN, J. (1990): *Systems Engineering - An Introduction*; Prentice Hall

BROOKS, F. (1987): "No Silver Bullets: Essence and Accidents of Software Engineering"; *Computer*, April 1987; pp.10-19

FAULK, S.R. (1997): "Software Requirements: A Tutorial"; pp.128-149 in DORFMAN, M. & THAYER,

R.H. (eds.) (1997): *Software Engineering*, IEEE Computer Society, California

GOUWS, J. & GOUWS, L.E. (2006): *Frameworks for Selected System Engineering Reports Ebook*; Feedforward Publications (http://payloadz.com/go/jump?id=268937&merch_id=8625&aff_id=9242); ESBN M01-175A-6885-77E1

KASSER, J. (1995): *Applying Total Quality Management to Systems Engineering*; Artech House

KAYTON, M. (1997): "A Practitioner's View of System Engineering"; *IEEE Trans. on Aerospace and Electronic Systems*, Vol.33, No.2, April 1997, pp.579-586

KERZNER, H. (1997): *Project Management: A Systems Approach to Planning, Scheduling, and Controlling* (Sixth Edition); John Wiley & Sons; ISBN 0471288357

LUTZ, R. (1993): "Analyzing Software Requirements Errors in Safety-Critical Embedded Systems", *Proc. IEEE Int'l Symposium on Requirements Engineering*, IEEE CS Press, Los Alamitos, California; pp.126-133

MIL-STD-490: *Specification Practices (Military Standard)*; Department of Defense, Washington, DC, USA

RHOADS, D.I. (editor) (1983): *System Engineering Management Guide*; Defense Systems Management College, Fort Belvoir, Virginia, USA

US-GAO (1992): *Mission Critical Systems: Defense Attempting to Address Major Software Challenges*; GAO/IMTEC-93-13; US General Accounting Office

VERMA, D. & FABRYCKY, W.J. (1997): "Systematically Identifying System Engineering Practices and Methods"; *IEEE Trans. on Aerospace and Electronic Systems*, Vol.AES-33, No.2, April 1997, pp.587-595

WYSS, S.E. (1988): "The ABCs of Specification Writing"; *Chemical Engineering*, May 9, 1988, pp.87-89

9. SELF-ASSESSMENT

9.1 TRUE / FALSE QUESTIONS

Indicate which of the following statements are TRUE and which are FALSE.

1. System level thinking is an essential skill of a professional engineer.
2. System engineering is only applicable in the defence-, aviation-, and space industries.
3. System engineering is a simple process.
4. The System Engineer always works closely with a Project Manager.
5. The system engineering process can only be executed sensibly if it is done randomly.
6. The development and utilisation of any complex engineering system typically take place in ten main life cycle phases.
7. Overlaps between the life cycle phases must be avoided in order to use resources sensibly.
8. The term *system* is often used generically, which can be very confusing, since one person's *system* can be several layers down in another's system hierarchy.
9. It is much better to get going with system development, instead of thinking too long about the

requirements. Requirements always become clear after the job had been done halfway.

10. Errors not fixed during the requirements phase can always be fixed later through feedback, and iteration.

9.2 MULTIPLE CHOICE QUESTIONS

Choose the one most correct answer for each of the following questions:

1) The *system engineering process* involves:
   a) Transforming a user requirement into a system specification
   b) Saving money
   c) Co-ordinate the acquisition of off-the-shelf components.
   d) Making sure the right software is purchased for systems in use.
2) Which of the following is not part of the system engineer's responsibilities?
   a) Addressing aspects such as interfacing, reliability, availability, maintainability, safety, survivability, producibility, and ergonomics.
   b) Providing help desk facilities for the organisation's computer users.
   c) Requirements definition and –analysis.
   d) System level design.
3) The most important components of a typical system specification are:
   a) Requirements formulation, and acquisition
   b) Utilisation and support, and phase-out and disposal.
   c) Both (a) and (b).
   d) None of the above.
4) Before system development can begin, it is necessary to:
   a) Use a top-down approach to organise the system into successive layers of functional units.
   b) Assign responsibility for development of the different units to different suppliers.
   c) Both (a) and (b).
   d) None of the above.
5) A *user requirements statement* (URS) is:
   a) A description of operational needs experienced by a specific user.
   b) A wish list, and therefore not very important.
   c) Nice to have, but not essential.
   d) The start of development, since it specifies the system to be developed.
6) The goal of a project's requirements definition phase is to:
   a) Decide who must do what.
   b) Decide what is possible and what not.
   c) Decide what to do, and to document the results of the decision-making process.
   d) Convince the client to be realistic.
7) Requirements can be classified as:
   a) Functional and non-functional.

b) Behavioural rituals and developmental quality circles.
c) Both (a) and (b).
d) None of the above.

8) The fact that requirements definition problems still exist today, in spite of extensive system engineering practices, indicates:
   a) That these practices are very limited in their scope.
   b) System engineering is a nice theoretical concept.
   c) The extent of systems is growing at a similar rate than system engineering practices.
   d) Further development and application of these practices are too costly to pursue.
9) Factors contributing to inherent difficulties in system development do not include:
   a) Comprehension.
   b) Uncertainty and communication.
   c) Inseparable concerns.
   d) Vague user requirements.
10) System development models include:
   a) XDM, YDM, ZDM.
   b) XDM, ADM, EDM, PPM.
   c) ADM, ATM, AM, PM.
   d) None of the above.

9.3 SHORT ESSAY QUESTIONS

(Typically these are typically 20 mark questions.)

1. Give your definition of system engineering.
2. Discuss why system engineering is important for the engineer, and what role problem analysis and rational decision-making can play in this regard.
3. Discuss the importance of proper requirements analysis, and how neglect in this regard can cause problems later on.
4. Discuss system hierarchies.
5. Discuss types of system specifications.
6. Discuss the typical system life-cycle for complex engineering systems.

****