

Chicagoland INCOSE chapter

# Systems Engineering Rules to Break

17 February 2011

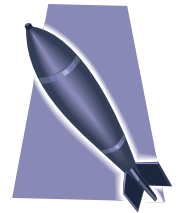
Steve J. Lindo  
CEO, SIM Solutions, Inc  
[sjlindo@scienceinman.com](mailto:sjlindo@scienceinman.com)

# Who Established the Rules?

- Publications
- Professional Organizations
  - INCOSE
  - IEEE
- Government Entities
- Industry Experts
  - Wasson, others
- Your Company
  - Corporate SOP's
  - Word of mouth

# Origins of Systems Engineering

- 1829 Rocket locomotive
- 1937 British air defense system team
- 1939-1945 Bell Labs supported NIKE dev.
- 1951-1980 SAGE Air Defense System – MIT
- 1956 RAND Corporation – Systems Analysis
- 1962 “A Methodology for Systems Engineering” published
- 1969 Modeling Urban Systems-MIT (Forrester)
- 1969 MIL-STD 499 Systems Engineering



# Origins of Systems Engineering

- 1990 INCOSE established
- By 1990, most rules were already established
- Common theme for prior rule-makers
  - Government sponsored
  - Big budgets
  - Prior to “Digital Revolution”
  - Not geared for commercial applications

# Why Rules Don't Always Work

- **The Theory:** Physics 101
  - Massless, frictionless pulleys
  - Projectiles neglecting air resistance
  - Electrical conductors without resistance
- **The Practice:** Physics in Real Life
  - Pulleys have friction
  - Projectiles have air resistance
  - Current-carrying wires heat up
- Systems Engineering rules work similarly!



# Writing Good Requirements

## □ Rules #1 & #2

- **#1: ALL REQUIREMENTS SHALL CONTAIN THE WORD “SHALL”**

- **#2: ALL REQUIREMENTS SHALL BE AUTONOMOUS**

- Purpose

- Distinguish binding requirements from design goals
- Convey which requirements must be verified and validated
- Allow the testing team to divvy up single requirements to various test writers without requiring any other context

# Writing Good Requirements

- When to break the “shall” and “autonomous” rules
  - ▣ Writing requirements for dynamic behavior
    - Example using autonomous, “shall” requirements

## 1.1 Startup Tests

1.1.1 Each time AC power is applied, the system shall perform a CRC test for memory integrity.

1.1.2 After the startup CRC test for memory integrity completes, the system shall write a block of 8 bytes to memory (Memory Write test).

1.1.3 After the startup Memory Write test completes, the system shall read back the bytes written in the Memory Write test (Memory Read test).

1.1.4 After the memory is read back in the Memory Read test, the system shall verify the block written in the Memory Write test matches the block read in the Memory Read test.

# Writing Good Requirements

## □ Example

- Dynamic behavior without using autonomous, “shall” requirements

### 1.1 Startup Tests Use Case

Precondition: AC power is applied.

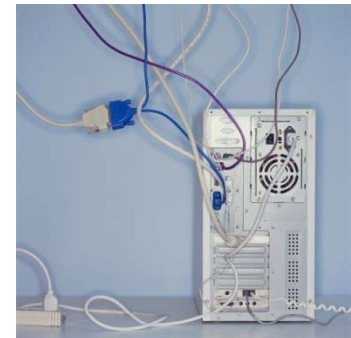
Dynamic Behavior:

The system performs a CRC memory integrity test.

The system writes 8 bytes to memory (Memory Write test).

The system reads back the 8 bytes written in the previous step (Memory Read test).

The system verifies the block written in the Memory Write test matches the block read in the Memory Read test.



# Writing Good Requirements

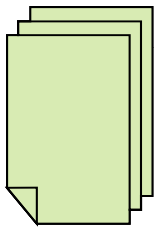
## □ Example

- How to designate mandatory requirements to V&V team without using “Shall” in each requirement? Use requirements mgmt system.

Mandatory?	Object Text
Header	1.1 Startup Tests Use Case
Precondition	AC power is applied.
Shall	The system performs a CRC memory integrity test.
Shall	The system writes 8 bytes to memory (Memory Write test).
Shall	The system reads back the 8 bytes written in the previous step (Memory Read test).
Shall	The system verifies the block written in the Memory Write test matches the block read in the Memory Read test.

# Writing Good Requirements

- ▣ Advantages of not using “shall” and autonomous requirements
  - Requirements written in plain English
  - More suited to dynamic requirements
  - Save paper
  - Easily managed in spreadsheet or Req Mgmt tool (i.e. DOORS)
  - If separate attribute/column maintained to designate mandatory requirements vs. design goals in an electronic requirements management system, it is easier to export or filter only the Shalls, only the design goals, etc.



# Writing Good Requirements

## □ Rule #3

- #3: Requirements must be singular

- Purpose:

- Ensure each requirement has at least one corresponding test case in a verification traceability matrix



# Writing Good Requirements

- When to break the singular rule
  - ▣ Large specifications often contain similar, related requirements hundreds of pages apart
  - ▣ Similar requirements, ideally, should appear together
  - ▣ A single compound requirement is easier for V&V team to manage than many related singular requirements spread out over many different sections of the requirements spec
  - ▣ Requirements which would be better suited in a short table or list

# Writing Good Requirements

## □ Example

### ▣ Using singular requirements

#### 1.2 Event Log

1.2.1 The system shall record unauthorized attempts to access the security settings to the event log.

1.2.2 The format for logging unauthorized attempts to access security settings shall be <UNAUTH><user ID> <space><MM/DD/YYYY><HH:MM:SS>

1.2.3 The system shall record UPS power activation events to the event log.

1.2.4 The format for logging UPS power activation events shall be <UPS\_ON><MM/DD/YYYY><HH:MM:SS>

# Writing Good Requirements

- Example

- ▣ Using compound requirements


## 1.2 Event Log

1.2.1 The system shall record the following events to the event log in the format specified:

Event	Format
Unauthorized attempts to access the security settings	<UNAUTH><user ID> <space><MM/DD/YYYY><HH:MM:SS>
UPS power activation	<UPS_ON><MM/DD/YYYY><HH:MM:SS>

# Writing Good Requirements

- How do you ensure a compound requirement is covered by the minimum number of test conditions?
  - Use a *Minimum # of Test Conditions* attribute/column in your electronic requirements management tool
  - DOORS example: Enforce minimum number of test case links corresponding to the *Minimum # Test Conditions*

Object Text	Minimum # Test Conditions
1.2 Event Log	0
1.2.1 The system shall record the following events to the event log in the format specified:	4 (4 In-links) 

# Writing Good Requirements

## □ Example

- Even singular requirements often have more than 1 minimum test condition (positive/negative testing), so a single link to a test case does not guarantee full test coverage.

Must ensure positive condition operates properly (5 cycles), and negative condition is not allowed (6 cycles)

Object Text	Minimum # Test Conditions
1.3 Programmable Limits	0
1.2.1 The system shall allow the user to program an upper limit of 5 therapy cycles	2 (2 In-links)

# Writing Good Requirements

## □ Rule #4

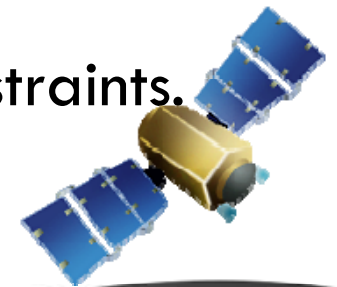
### □ #4 Don't specify the design in the requirements

#### □ Purpose

- Allow designers the flexibility to come up with a better approach to achieve the desired outcome—don't over-constrain.

# Writing Good Requirements

- When to break the “don’t specify the design” rule
  - All requirements are the result of design decisions
  - Once the architecture has been established, and feasibility or proof of concept models support the architecture, feel free to specify design in the lower-level subsystem specs, while leaving the high-level functional and performance requirements design-neutral.
  - When interfaces are defined by external constraints.



# Writing Good Requirements

## □ Example

- System-level requirement: The aircraft's climb rate from sea level to 30,000 feet shall be no less than 20 minutes.
- Turboprop concept chosen, based on cost.
- Create model to calculate horsepower required to achieve this climb rate, assuming reasonable gross weight, propeller efficiency, and a 1% Mil-Std Hot Day
- Engine subsystem requirement: The engine shall have a shaft horsepower of no less than 455 HP at sea level.

# SE Process

## □ Rule #5

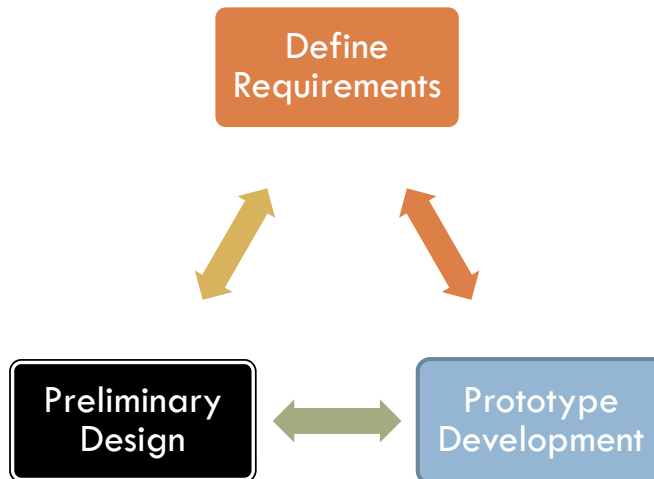
□ **#5: You must define all the requirements first, then go design/build it.**

□ Purpose:

- Spend a small amount up front defining the requirements before spending a large amount reworking and redesigning it later

# SE Process

- When to break the “define requirements before designing / building it” rule
  - ▣ Always. Requirements are developed iteratively.
  - ▣ Stay a few steps ahead of the designers.
  - ▣ Be open to requirements changes driven by design activities.



# SE Process

- When to break the “define requirements before designing / building it” rule
  - ▣ If prototyping and proof of concept models will help steer the system architecture
  - ▣ If management or customer is likely to base funding decisions for later stage development on seeing some tangible product they can hold
  - ▣ If speed to market is more important than \$ spent
  - ▣ Risk: Design is not done until the requirements are defined and verified/validated



# Defining User Needs

## □ Rule #6

- **#6: “Elicit Stakeholder Requirements from stakeholders who will have an interest in the system throughout its entire life cycle. (INCOSE Handbook v3.2)**
- **Purpose**
  - Ensure all relevant stakeholders’ interests are met in the design, including end user, service representative, interfacing systems, etc., to prevent future rework if these needs aren’t discovered until after the system is built.

# Defining User Needs

- When to break the “elicit stakeholder requirements” rule
  - ▣ When the stakeholder paying for the system isn’t willing to pay extra for the features
    - Conjoint analysis—good tool to determine this
  - ▣ When the technical risk outweighs the benefit of the added features (weight, reliability, speed to market, etc.)

# Defining User Needs

## □ Example

- Medical device, paid for by Medicare, operated by patients
- Patients describe their user needs in terms of nice-to-have items, such as voice-activated prompts
- Medicare's needs are cost-driven, with a fixed reimbursement for this type of device
- SE may write up the requirement for voice-activated prompts, which adds cost that the “other” customer (payer) isn't willing to incur.

# Risk Management

## □ Rule #7

□ #7: Risk Acceptability Matrix is a good tool for ensuring residual risk is reasonable

## □ Purpose

### ■ FMEA

- Identify failure modes
- Identify effects of failure modes
- Estimate probability of occurrence of failure
- Estimate severity of harm due to failure

■ Use a risk acceptability table to determine if a failure mode requires additional risk controls

# Risk Management

- When to break the Risk Acceptability Rule
  - ▣ Always. Sum up overall probability of occurrence of each category of harm instead of relying on Acceptability table.

	Occurrence				
Severity	Not likely	Seldom	Sometimes	Occasional	Frequent
No harm	ACCEPTABLE	ACCEPTABLE	ACCEPTABLE	ALARP	ALARP
Minor harm	ACCEPTABLE	ACCEPTABLE	ALARP	ALARP	UNACCEPTABLE
Intermediate harm	ACCEPTABLE	ALARP	ALARP	UNACCEPTABLE	UNACCEPTABLE
Major harm	ALARP	ALARP	UNACCEPTABLE	UNACCEPTABLE	UNACCEPTABLE
Death	ALARP	ALARP	UNACCEPTABLE	UNACCEPTABLE	UNACCEPTABLE

# Risk Management

## □ Example

□ Single FMEA line item shown below:

Hazard	Occurrence	Harm	Acceptability
Damaged power cord	Occasional	Intermediate (Shock)	Unacceptable

□ Team splits this into 2 FMEA items as follows

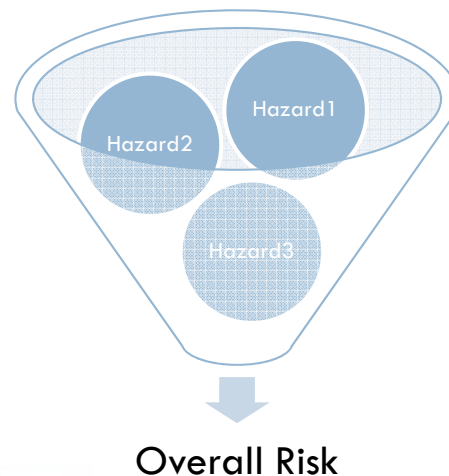
Hazard	Occurrence	Harm	Acceptability
Damaged power cord (worn insulation)	Sometimes	Intermediate (shock)	ALARP
Damaged power cord (wires pulled from housing)	Sometimes	Intermediate (shock)	ALARP

□ Risk now acceptable without adding mitigations!

# Risk Management

## □ Example

- Use quantitative values whenever possible
- Estimate overall likelihood of the hazardous situation leading to each type of harm
  - In previous example, sum up overall probability of shock-related items instead



# Your Experiences

- What rules have you broken successfully?
- What rules appear to be outdated?
- What rules are missing from common publications and literature on SE best practices?

# References

- INCOSE Systems Engineering Handbook v. 3.2, INCOSE-TP-2003-002-03.2, January 2010

# Questions and Discussion

Steve J. Lindo

CEO, founder, SIM Solutions, Inc.

Director at Large, INCOSE Chicagoland Chapter

214-557-5868 cell

773-283-4500 fax

[sjlindo@scienceinman.com](mailto:sjlindo@scienceinman.com)

Science . Innovation . Management