Friends, Romans, countrymen use your EARS & Improve your requirements

(Not from Julius Caesar by William Shakespeare –)

Introduction

I Work for Siemens within the Rail Automation business.

We are based in Ashby de la Zouch right in the middle of England.

What we do.

Design, build and install systems that monitor, control and report on rail network infrastructure.

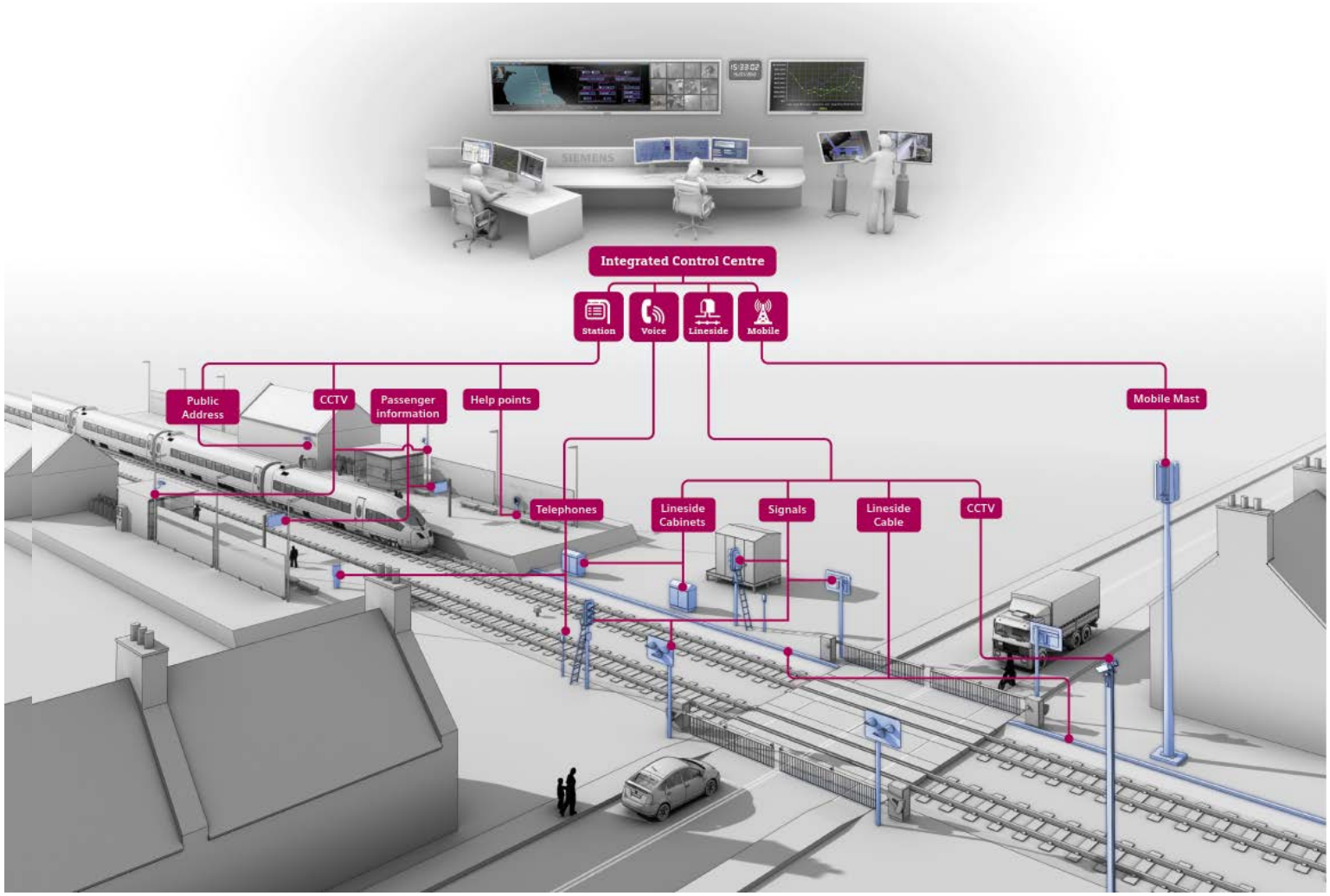
Inform passengers of arrivals, timetable changes etc via PA and electronic signs.

Keeping passengers safe with CCTV monitor, help points etc

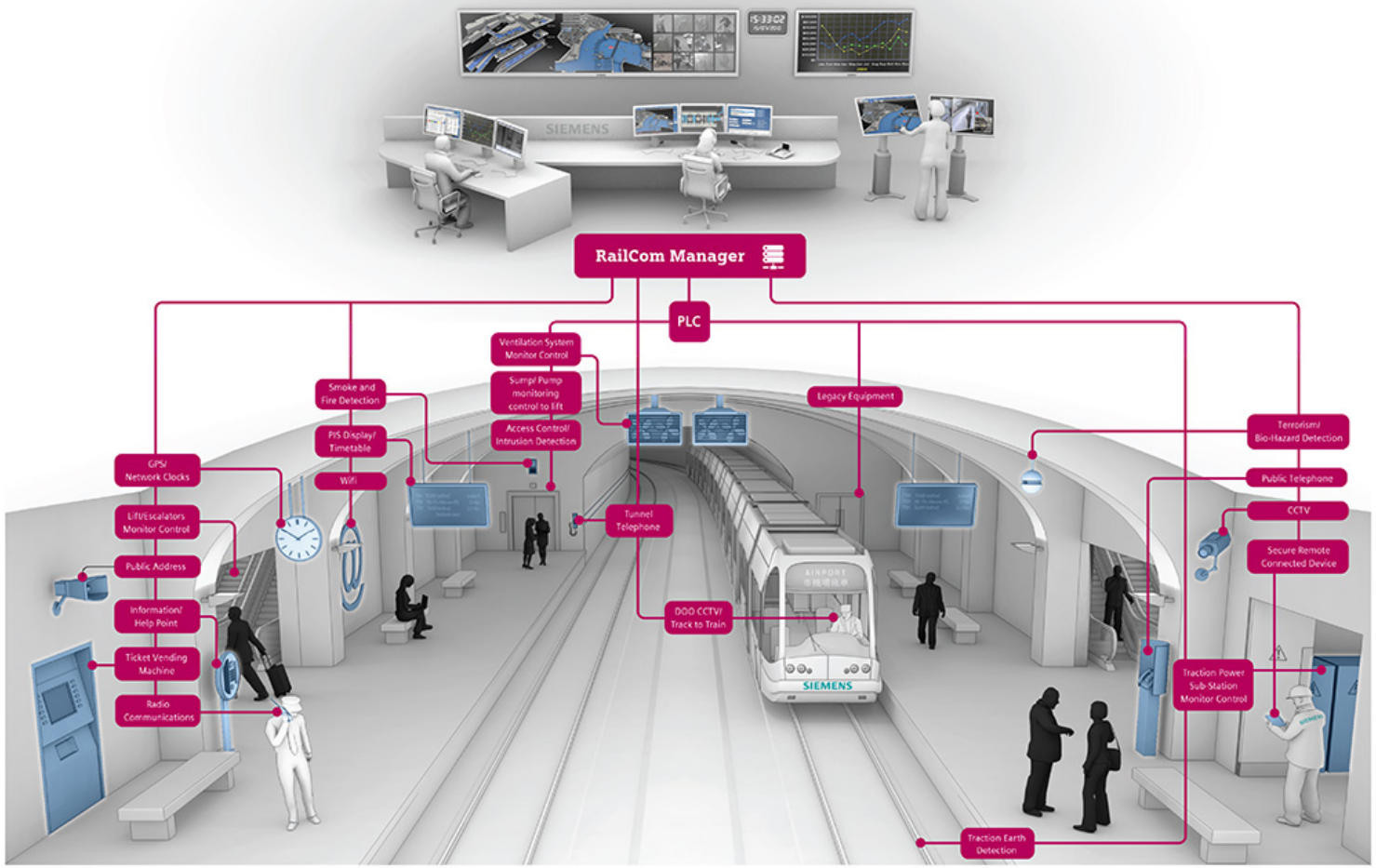
Enabling operators to monitor and manage trackside assets to both warn of and predict failures.

Secure control and monitoring of traction power including all switches, circuit breakers and rectifiers.

What we do.



What we do.



Challenges

In common with many growing business we face some challenges.

Systems are getting larger, more complex with more interfaces.

Need to modernise some parts of our architecture whilst still responding quickly to our customer needs.

Desire to reuse more assets.

Regulated Industry with increasing governance requirements.

Initiatives

On going improvement initiatives highlighted the need to embrace some fresh techniques and methods.

Earlier this year we adopted Polarion & PlasticSCM to support these initiatives, to form a platform for rolling them out and acting as a the engineering “system of record”.

One of our first initiatives looked at the way we express system & software requirements.

This forms the basis for this presentation.

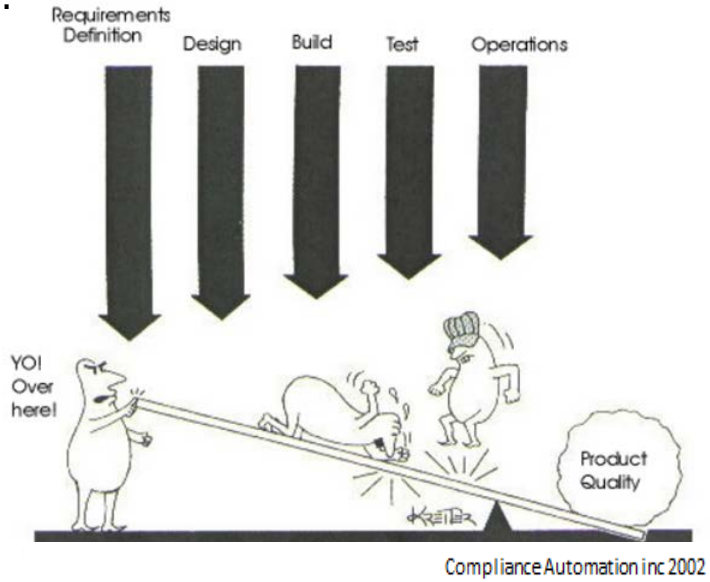
What do we know about requirements?

Requirements are engineering's first opportunity to fail.

Significant evidence that ‘poor’ requirements are the number one cause of project failure: (late, over budget, cancelled, not fit for purpose).

‘Good’ Requirements can act as a lever to:

- Improving system quality & development success.
- Respond better to customer needs.
- Help managing complexity.
- Facilitating reuse.



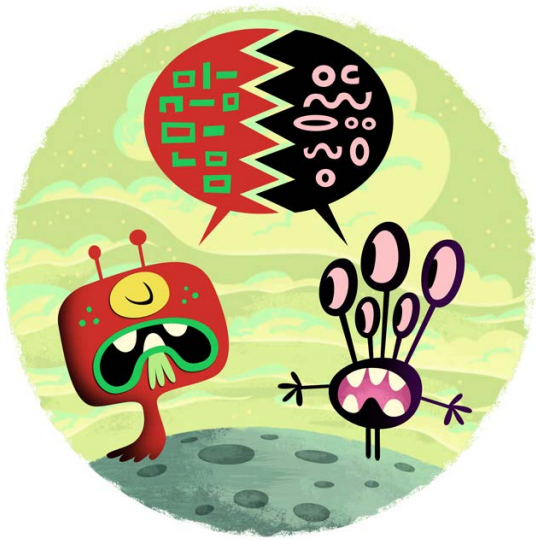
What's wrong with requirements?

The most popular way of expressing system requirements is in natural language captured using text.

This can make it hard to be clear, precise & unambiguous but its available to all, readily understood and allows almost any concept to be expressed.

Alternative exist such as SysML & Planguage but these are also imperfect.

- They require training and take time to master.
- Can limit what can be expressed.
- May be hard for customers to understand.
- May present issues with end to end trace.



How to identify ‘Good’ Requirements?

INCOSE* identify 9 characteristics of good requirements :

- Necessary
- Implementation Independent
- Unambiguous
- Complete
- Singular
- Feasible
- Verifiable
- Correct
- Conforming

In evaluating our existing requirements we noticed the following issues:

- Ambiguous
- Verbose – wordy & complex
- Omissions
- Not testable/ verifiable
- Duplication

*INCOSE TP 2010-006-001 Guide to writing requirements

SIEMENS

User Stories

Agile software development favours **User Stories** why can't these be used for system requirements?

User story format :

As a <type of user>, I want <goal> so that <reason>

Example:

As a train operator, I want low cost rolling stock so that I make a profit.

This appears to capture who wants what and why.

I might quibble about what 'low cost' means and there is some vagueness and ambiguity.

Could we pass this on to a subcontractor to work against?

User Stories

As a train operator, I want low cost rolling stock so that I make a profit.

Maybe not!

SIEMENS

The rail industry is heavily regulated and this places additional demands any implementation.

The systems we deliver are composed of many systems (a system of systems). We need a way of expressing requirements that can be understood by the customer, by us and used for subsystem selection and acceptance.

User Stories look ideal for capturing the essence of a need, its stakeholders and why it is needed, but we need something :

- more precise.
- that helps remove ambiguity
- that is verifiable.
- that can expand on a user story.
- That reduces our risk and adds value.

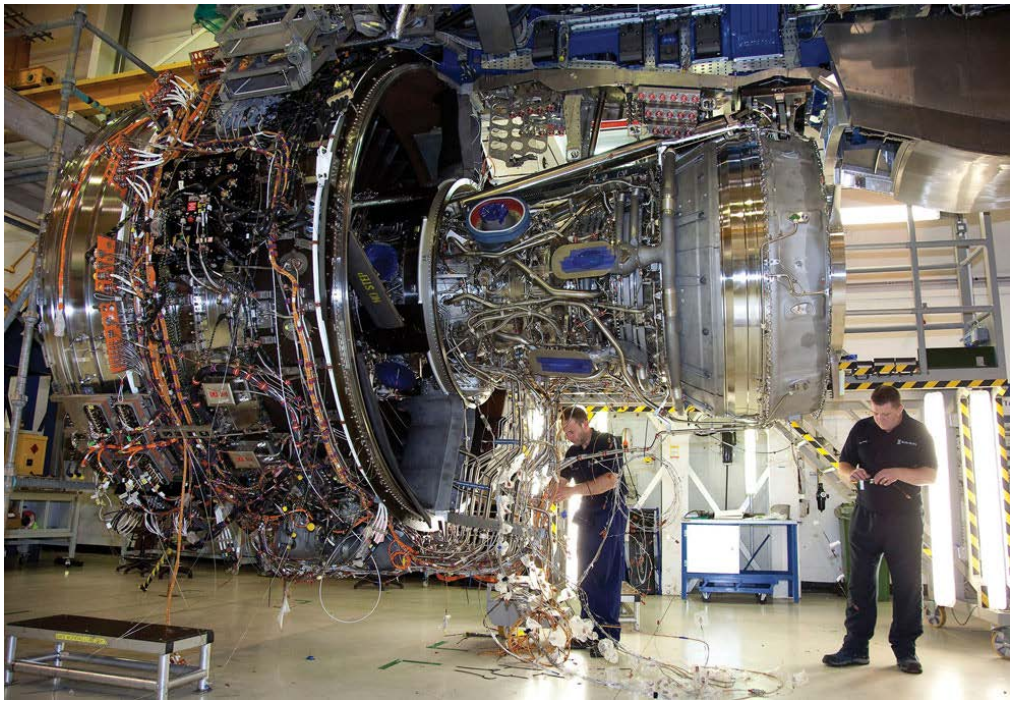
EARS - Easy Approach to Requirements Specification.

SIEMENS

Developed by Alistair Mavin et al, at Roll-Royce.

Not proprietary.

Has been used to express the functional requirements of an aircraft engine control system. These are safety critical, span several disciplines (HW, SW, Mechanical) and are complex.



SIEMENS

EARS

EARS describes what the system shall do as a black box. What it shall do at the identified boundary of the system.

EARS limits the use of natural language by guiding authors towards 1 of 5 basic templates:

- 3 are used to capture **normal** behaviour.
- 1 is used to express **unwanted** behaviour.
- 1 is used to capture **optional** features.

Each template has a simple syntax constructed from simple clauses.

Each clauses appears in a specific order.

Each template has compulsory and optional clauses.

Each template begins with a keyword identifying its type.

SIEMENS

EARS – Four Types of Normal Behaviour

Normal behaviour describes what a system should do.

EARS divides normal behaviour into 4 possible types:

Constant (Ubiquitous)	A behaviour that is constant, a fundamental system property that requires no stimulus.
Event-Driven	A behaviour that is initiated **when** a trigger occurs or is detected.
State-Driven	A behaviour that exists **while** the system is in a given state or mode.
Complex	A combination of the above for example behaviour **while** the system is in a specific mode and **when** an event arrives.

EARS – Constant Template (AKA Ubiquitous)

Syntax:

The <system name>, **shall** <system response>

Example(s):

The PA system speakers, **shall** be located as detailed in the acoustic survey.

The Phone, **shall** be fitted with one type-c USB connector.

Notes:

Valid Constant requirements usually detail a fundamental system property.

Things that at first appear **Constant** often aren't!

The passenger help point, **shall** call the passenger help desk. – "Really"

EARS – Event-Driven Template

Syntax:

When <trigger >, **the** <system name>, **shall** <system response>

Example(s):

When activated, the help point, **shall** call the passenger help desk.

When a higher priority announcement is requested, **the** PA system, **shall** abort any in progress announcement and start the higher priority announcement.

When USB cable connected, the phone, **shall** enter charging mode.

Notes:

Event-driven requirements are activate only **when** the trigger occurs. Triggers come from outside the system boundary.

EARS – State-Driven Template

Syntax:

While <pre-condition>, **the** <system name>, **shall** <system response>

Example(s):

While Fire Alarm in progress, **the** PA system, **shall** permit only emergency announcements.

While in low power mode, **the** phone software **shall** reduce screen brightness by 20%.

Notes:

State-Driven requirements are activate only **While** in the specified state/ mode.

EARS – Complex Template

Syntax:

While <pre-condition(s)>, **When** <trigger>, the <system name> **shall** <system response>.

Example(s):

While nighttime reduced volume active, and fire alarm active, **When** PA microphone talk button pushed, the PA system **shall** suspend nighttime reduced volume.

While powering up, **When** the phone software detects an external memory card, the phone software shall use the external card to store all user data.

Notes:

Describes **Complex** requirements involving many triggers, states and/or other EARS template clauses.

SIEMENS

EARS – Unwanted Behaviour

EARS provides a specific template for unwanted behaviour such as error conditions and faults.

It's a variation of the Event-Driven syntax where the event is the detection or occurrence of the unwanted behaviour.

Uses the keyword **IF** to distinguish unwanted behaviour from wanted.

Defines the circumstance and the required response to the unwanted situation.

May be combined with other templates to specify more **complex** unwanted behaviours.

EARS – Unwanted Behaviour Template

Syntax:

If <trigger>, **then** the <system name> **shall** <system response>

Example(s):

If checksum invalid, **then** software shall request a message resend.

If notice badly formatted for chosen passenger information display **then** the system **shall** display “format error” message to user.

Notes:

Can be combined with other templates to form a **Complex** unwanted requirements.

EARS – Optional Feature Template

EARS provides a template to express functionality when optional features or components are included.

Allows simple system variability to be expressed.

Permits system(s) to be expressed with optional functionality from the start.

EARS – Optional Feature Template

Syntax:

Where <feature name>,**the** <system name> **shall** <system response>

Example(s):

Where multiple languages available, **the** Information Display **shall** cycle through them display each for 10 seconds.

Where zooming CCTV camera installed **the** software **shall** enable the zoom menu and associated functionality.

Notes:

Typically Optional features will be **Constant** for systems that have them fitted.

Optional subsystem would then have its own EARS requirements.

SIEMENS

EARS – All Templates

Constant/ Ubiquitous	**The <system name> shall <system response>** *The PAS shall allow operator recorded messages to be scheduled.*
Event Driven	**When* <trigger>, the <system> shall <system response>** *When higher priority announcement requested, the PAS shall interrupt any lower priority announcement in progress.*
State Driven	**While <in state>, the <system> shall <system response>** *While fire alarm sounding, the PAS shall permit only fire announcements.*
Unwanted	**IF* <trigger>, then the <system> shall <system response>** *If power fails then the system shall remain full operational for at least 1 hour.*
Optional	**Where <feature>, the <system> shall <system response>** *Where multiple languages, the PID shall cycle between them displaying each for 30 seconds.*

SIEMENS

Applying EARS

EARS is deceptively simple.

In attempting to reword a requirement using an EARS template many questions will be raised and need answering.

- Is there a trigger? What is it? Where does it come from? How will it arrive?..
- Is there a mode? What is it?...

These can be answered in collaboration with customers and stakeholders as EARS can easily be explained and understood by stakeholders.

The result is an agreed set of requirements that are more easily verified and that can form part of an acceptance case.

Applying EARS

Pros:

EARS requires limited training, users go from novice to proficient quickly.

Having a common format simplifies communication and everyone to recognise a good requirement.

EARS templates reduces ambiguity, duplicate and conflicting requirements.

Requirements reviews now concentrate on the requirement and not natural language issues such as grammar.

Weaknesses:

Limited inter-requirement coupling.

Not suitable for very complex requirements – EARS+ should help here.

SIEMENS

A Final EARS Example.

This example from an Intel paper shows what happens when EARS patterns are applied .

Original:

The software shall warn the user of low battery.

Rewritten using EARS:

While on battery power, if the battery charge falls below 10% remaining, **then** the software shall display a warning message requesting a switch to AC power.

What low power means is quantified.

A message that needs to be displayed is now identified.

As is what is needs to say.

SIEMENS

EARS and Polarion

Whilst EARS requires no specific support from a tool its use can be enhanced by one.

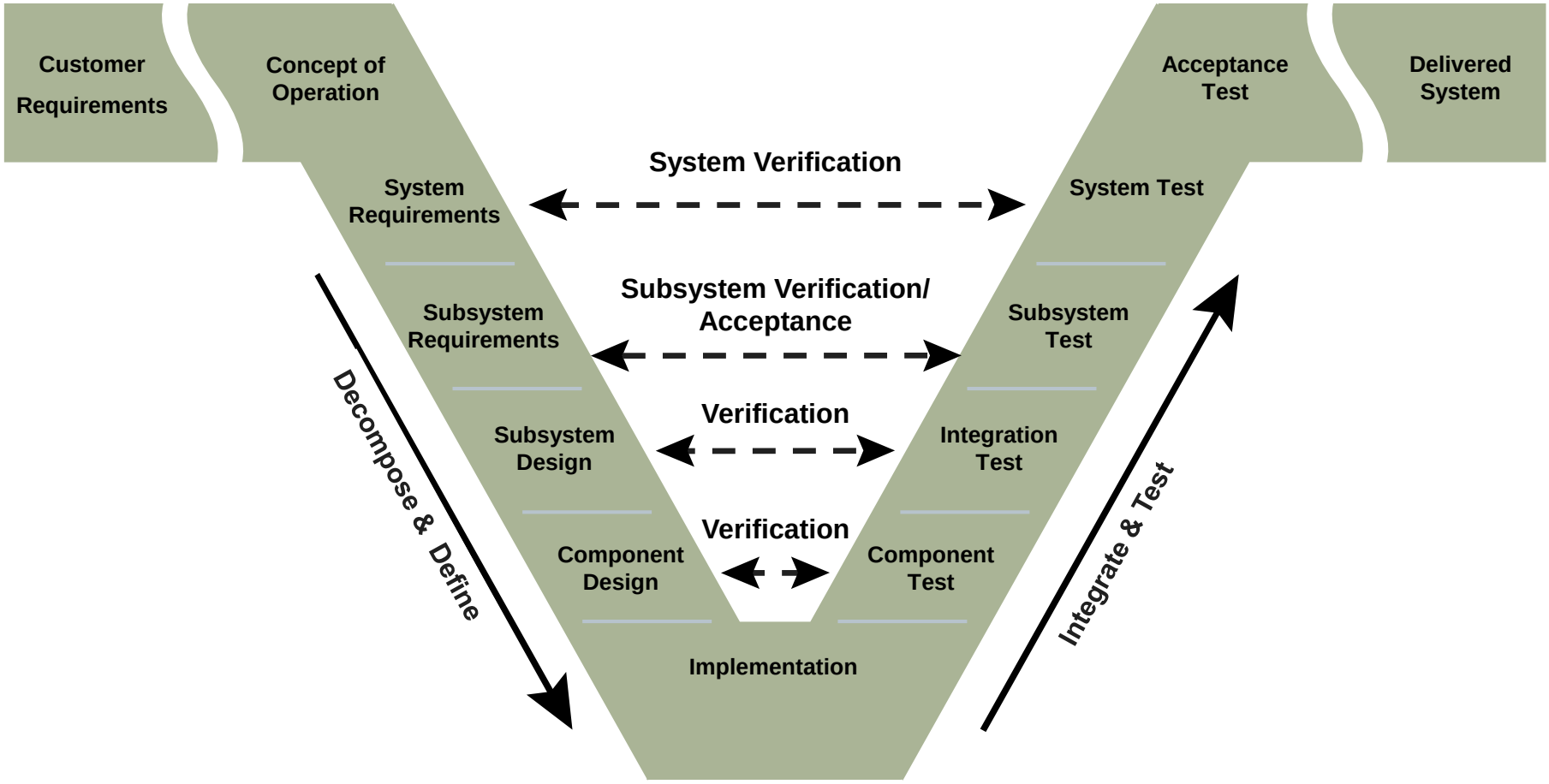
Polarion allows requirements to be created and managed at many levels of granularity (SPACES) with easy traceability.

We sometimes get requirements from customers. These can be of variable quality.

We also get requirements from internal stokeholds such as sales and from road mapping exercises. These may be high level and incomplete. User Stories might be good here.

SIEMENS

Conceptual Engineering V'



SIEMENS

Customer & Stakeholder Requirements

Requirements of variable quality and completeness can be recorded in Polarion in their own “Customer” Space

SIEMENS

We can refine these into EARS format as we move down the left hand side of the conceptual V and trace from the Customer space to a (system) Requirements Space

We can also use verification traces.

SIEMENS

in due course we may use the optional **where** requirements as the starting of a product feature model and move toward a product line approach.

The new (optional) feature model and variant management features of Polarion would enhance this.

Questions?

Any Questions?

SIEMENS

Acknowledgments

Mavin, A., Wilkinson, P., Harwood, A., and Novak, M.;

"EARS (Easy Approach to Requirements Syntax)", Proceedings 17th IEEE International Requirements Engineering Conference IEEE, pp. 317-322

EARS+ is Scheduled to be published later this year. It allows EARS templates to be further refined with additional clauses that provide more options for those requirements that need it.

Contact Details

Dale Gillibrand
Specialist Engineer.
Siemens Rail Automation,
Ashby Park,
Ashby de la Zouch,
Leicestershire,
LE65 1JD,
United Kingdom.

E-mail:
dale.gillibrand@siemens.com
Linked-In

siemens.co.uk

Extra

Three layer development Model