

10 Simple Steps to Better Requirements

An Industry Track Tutorial
at RE'06

Ian Alexander

ian@scenarioplus.org.uk



Writing Better Requirements

*Fare il buon vino e semplice,
ma non facile.*

*An Italian peasant farmer,
to Richard Stevens.*

Making good wine is simple,
but not easy.*

** Or requirements!*

Contents

- Mission and Scope
- Stakeholders
- Goals & Goal Conflicts
- Dictionary
- Scenarios & Exceptions
- Requirements
- Justifications
- Assumptions
- Agreed Priorities
- Acceptance Criteria

Mission and Scope



Identify Business Objectives

- First stage in requirements is to get a simple statement of purpose
- Business Objectives may be broader than System Scope
 - many processes will remain un-automated
- Objectives must not conflict with each other
- Overall Timescale and Cost may be vital clues

Objectives Example

- “To become the market leader in small-household burglar alarms”
- “To make steadily growing annual income from alarm sales, maintenance, and monitoring”

Must not conflict!

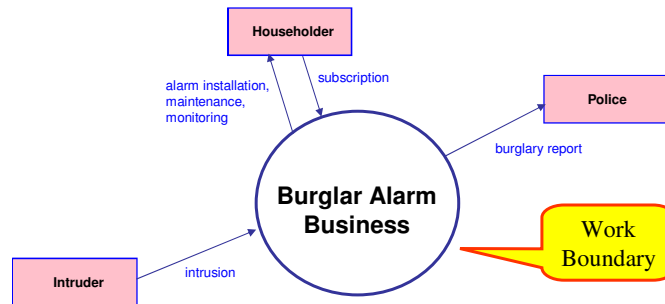
Clear, Simple, Truthful –
with many implied requirements



Mission & Objectives Template on Website

Scope

- Agree what is (and what isn't) in Scope



- ⇒ Draw a Context Diagram for the whole of the Work to be done
- ⇒ Must be consistent with the Business Objectives

© Ian Alexander 2006

7

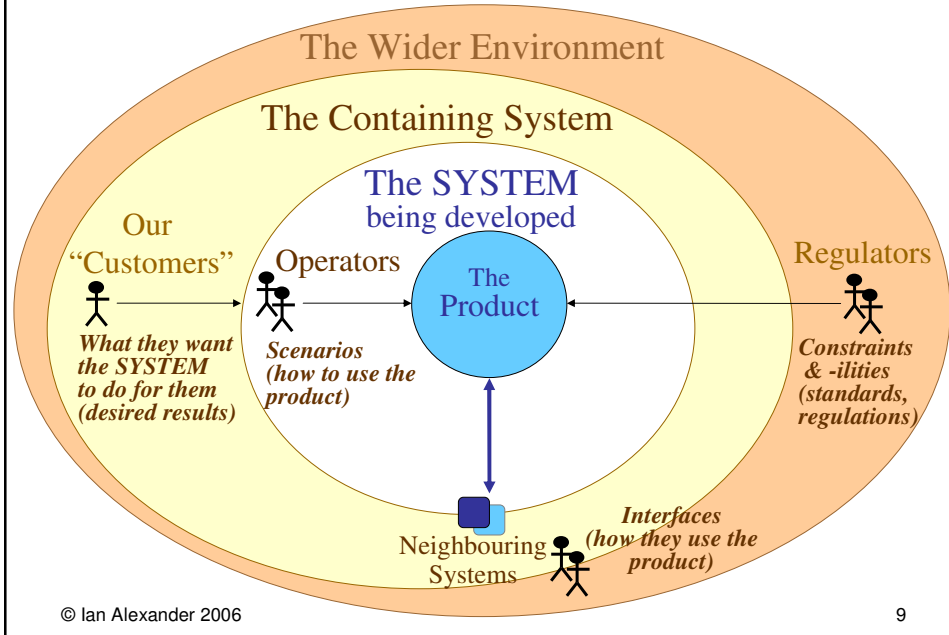
Stakeholders



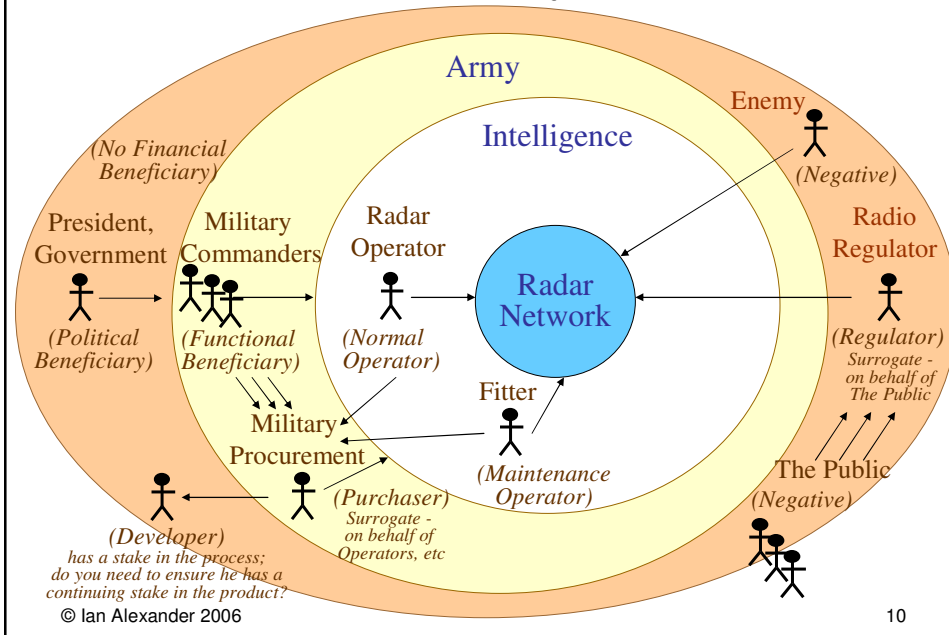
© Ian Alexander 2006

8

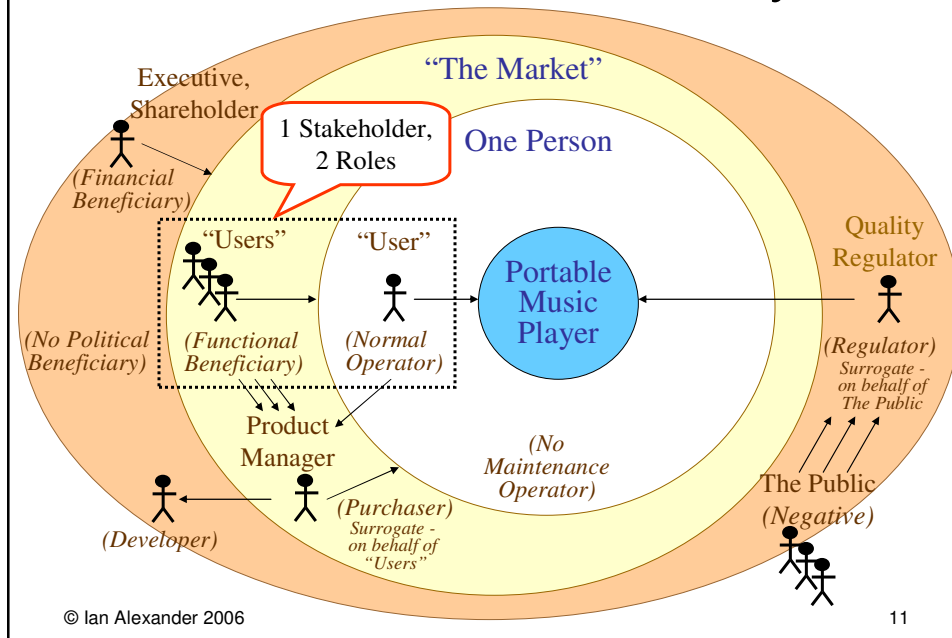
Roles in the System Hierarchy



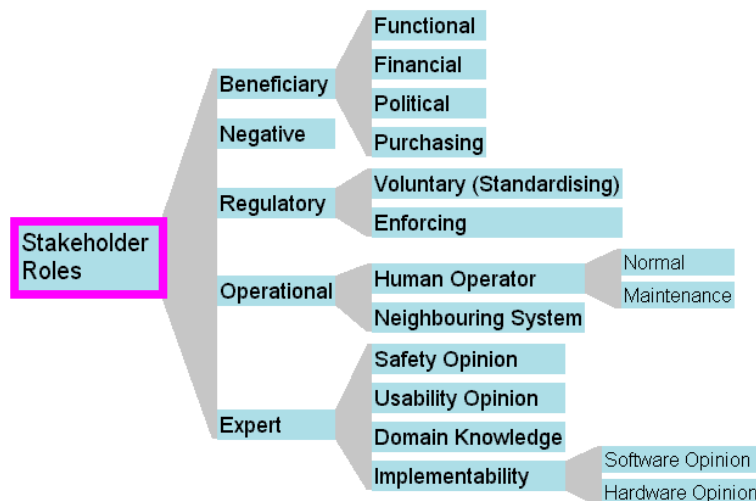
Roles: Military Radar



Roles: Portable Music Player



A Template for Stakeholder Roles



 **Stakeholders** Template on Website

© Ian Alexander 2006

12

Belbin's Team Roles

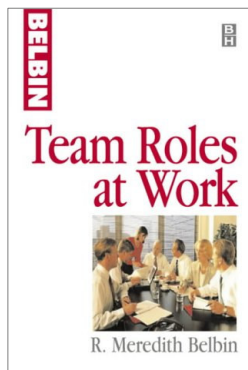
- People vary, but in a predictable way
 - Belbin describes 9 Team Roles that people typically play (1 or more of).
- A happy project
 - has a good mix of people, who together can cover all the project stakeholder roles.
- A happy project worker
 - is a person whose current **Project Role(s)** is/are a good match for their personal Belbin **Team Roles**.

© Ian Alexander 2006

13



'Team Roles'



Meredith Belbin
Team Roles at Work
Butterworth-Heinemann 1996

Key Concepts:

- people vary, but can be understood as playing any mixture of 9 different team roles. You need a balance on every project.

Strengths:

- Useful for understanding team dynamics.

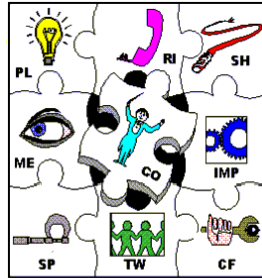
Weaknesses:

- Other role classifications exist.

© Ian Alexander 2006

14

Belbin's Team Roles



BELBIN TEAM ROLES

www.belbin.info

Team Role	What this Role does well	Weaknesses you should forgive in this Role
Plant	Goes deeply into problems, invents, imagines new solutions, creative, unorthodox	Not good at details May be too distracted to communicate well
Resource Investigator	Loves communicating, getting things going with people	Over-optimistic Losing interest after a while
Shaper	Vision, shaping the project, pushing to overcome barriers	Annoying other people
Monitor Evaluator	Making accurate judgements	Lacking drive Too critical
Coordinator	Being a confident chairperson, making decisions	Manipulating people
Implementer	Organised, efficient, sees to details, reliable	A bit inflexible Slow to change
Specialist	Single-minded, seeking knowledge	Only contributing on a narrow front
Teamworker	Co-operating, being calm and diplomatic, listening	Indecisive Easily influenced
Completer Finisher	Conscientious, painstaking	Reluctant to delegate

© Ian Alexander 2006

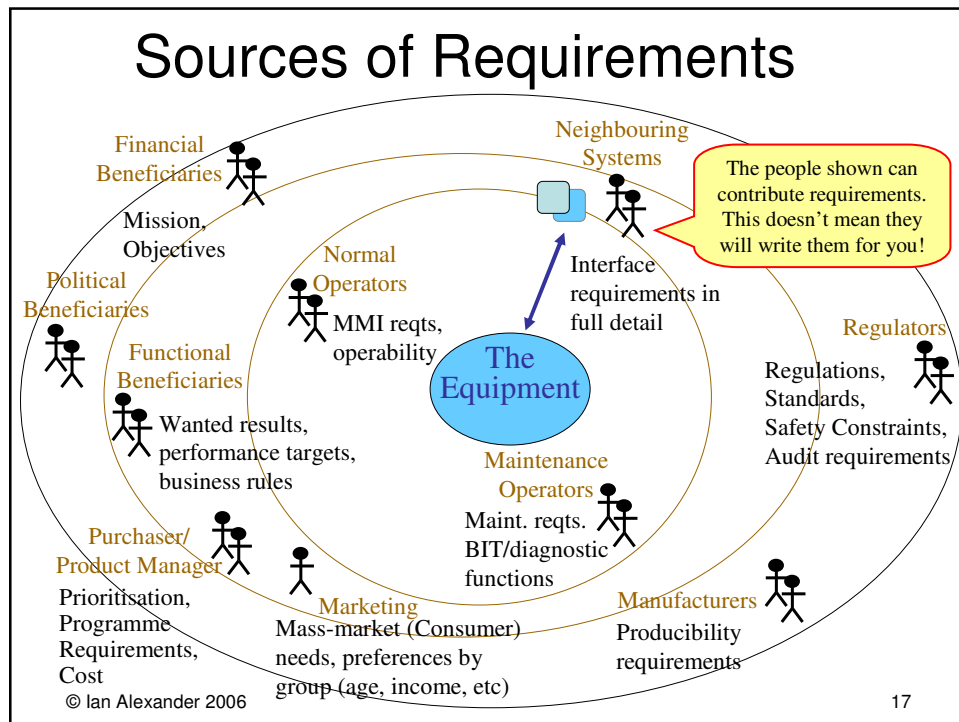
15

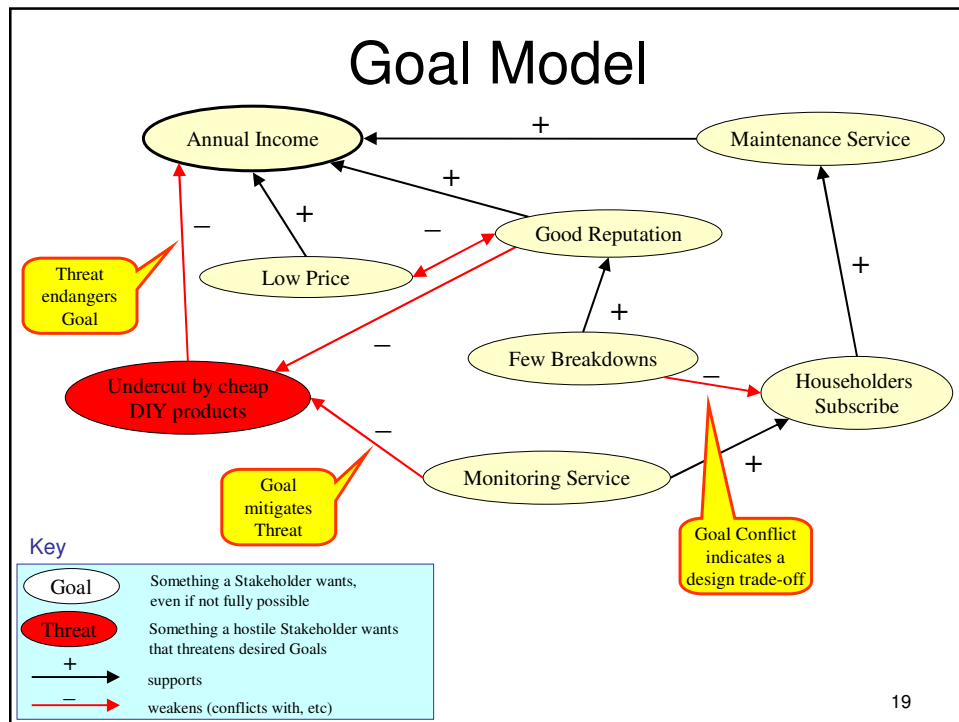
How does Stakeholder Analysis Help?

- You know who to listen to
- You know which kinds of requirement you need to get...
- ...and who can help you with them
- You don't miss out whole chunks of the specifications.

© Ian Alexander 2006

16





Goals are not Requirements

Goals	Requirements
may conflict, indicating trade-offs	must not conflict
may be ideal, unattainable, indicating what is hoped for	must be realisable within limits of budget, timescale, technology, and skill available

“A perfectly safe railway network”

© Ian Alexander 2006

20

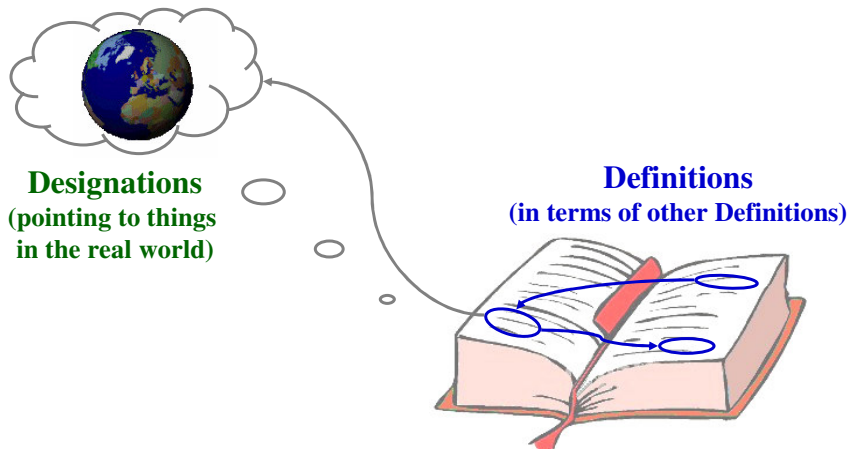
Dictionary



© Ian Alexander 2006

21

Dictionary: Definitions v. Designations



- Provide the bare minimum of links to the outside world
- Define all other terms consistently
 - Using these links to the outside world
 - But mostly in relation to each other

© Ian Alexander 2006

22

Can you rely on (designated) 'things in the world'?

- Does your Product ever really 'know' what's happening in the world?
- Or only about quantities (variables) provided by its own interfaces – sensors and the like?
- Conclusion? Designated things are not too reliable. You can't define them fully, and you can't ever exactly measure them, either

© Ian Alexander 2006

23

Making it Clear: Project Dictionary

Suspected Intrusion:	Message from Household Alarm to Control Centre , indicating a possible Intrusion for Confirmation
-----------------------------	---

- Different domains usually have their own languages
- Common for different people to use overlapping terms, synonyms, vague terms, even obviously-wrong terms
- Dictionary is essential to resolve confusion
- Dictionary provides a single, project wide, agreed set of terms and their definitions

⇒ *Minimise (undefinable) references to outside reality*

⇒ *Check for 'closure' – that all terms used are defined
(or at least referred to outside reality)*

© Ian Alexander 2006

24

Scenarios



© Ian Alexander 2006

25

Writing a Scenario

Operate the Train

- 1 At the depot, Train Maintainer **prepares the train**, often overnight, and makes it available for use.
- 2 Train Operator **takes train into service** to carry passengers.
- 3 Train Operator stops at the first station encountered and changes to Auto mode, which connects the Auto Driver Box with motoring and braking.
- 4 The Auto Driver Box **operates train in auto**.

Goal: say what the Scenario does, in a short active phrase

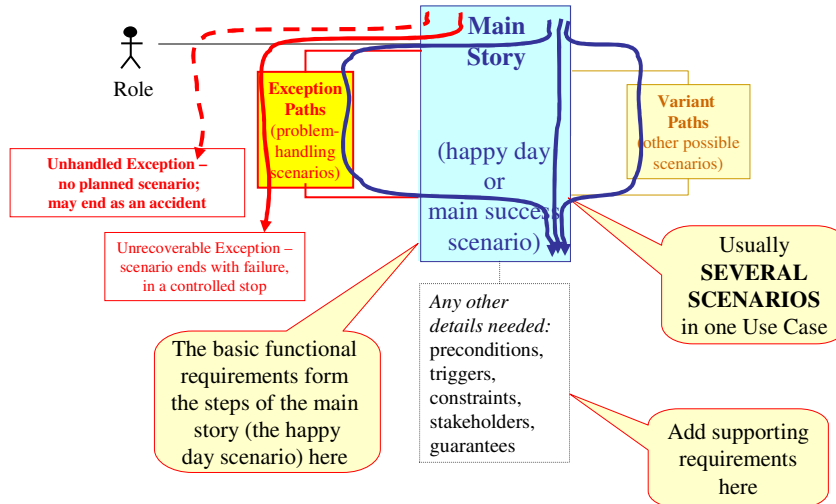
Story: tell the story as a simple list of who-does-what steps:
<Role> does <Action>

Details: put any complicated sub-story in a Scenario of its own, and **include** its name

© Ian Alexander 2006

26

Use Case: One Goal, Several Scenarios

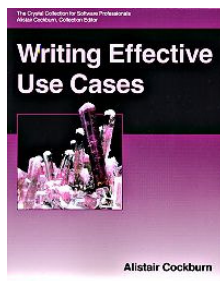


Ask yourself – am I telling the story clearly?

© Ian Alexander 2006

27

Use Cases (a way of organising Scenarios)



Alistair Cockburn
Writing Effective Use Cases
Addison-Wesley 2001

Key Concepts:

- Use Cases are chunks of behaviour
- They can be nested to form a Goal (AND/OR) tree
- Exceptions branch off from normal paths
- Roles, Triggers, Preconditions, Guarantees

Strengths:

- Narrative makes behaviour easy to understand, and to check

Weaknesses:

- Best for episodic behaviour, e.g. event responses; not ideal for everything (weather forecasting?)

© Ian Alexander 2006

28

Use Case Example

Protect the House

Primary Scenario

HH activates Alarm.
 HH closes door.
 Alarm listens out on all its sensors for possible [Detected Intrusion](#).
 HH opens door.
 HH deactivates alarm.

Each step is a possible functional requirement

Variant Paths

HH activates Alarm from outside the house (deluxe system only).

Exceptions

Alarm cannot be activated: HH contacts Control Centre.

Guarantees

Alarm indicates to HH when it is active/inactive.
 Alarm does not activate if faulty.

Non-functional requirements that apply in this specific case

Local NFRs

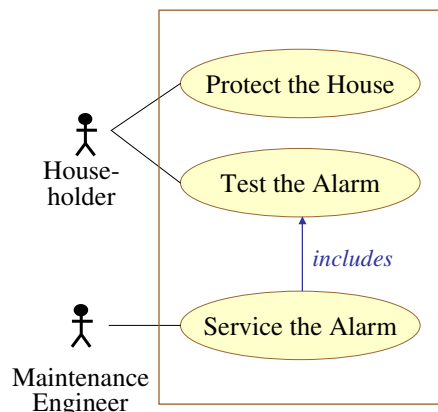
Alarm is usable by householder with 5 minutes training.

© Ian Alexander 2006

29

Use Case Summary Diagram

- Identifies ways that Roles interact with a System
- Vertical list does not imply sequence
(but people will tend to interpret it as such!)
- Does not say what is in the Use Cases, it's only a graphical index
- Most types of project need to do more to define system Context (*see later*)



© Ian Alexander 2006

30

Use Case Document Structure

Introduction

- Business Objectives
- Stakeholders (Operational/not)
- References
- Conventions Used

Use Cases

- Group₁ (*Cases in Use Case Diagram₁*)
 - Use Case₁
 - Primary Scenario (*steps 1, 2, 3...*)
 - Variant Paths
 - Exceptions
 - Local NFRs
 - Next Use Case...
- Next Use Case Group...

Global NFRs (*see NFR Template*)



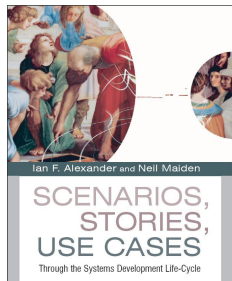
Use the *Use Cases* Templates (they range from sketchy to detailed)

© Ian Alexander 2006

31



Scenarios



Ian Alexander & Neil Maiden
Scenarios, Stories, Use Cases
through the systems development life-cycle
John Wiley, 2004

Key Concepts:

- Scenarios are starting point for all specification and design
- Scenarios help with process modelling, requirement gathering, specification, design, testing, operations, maintenance, and training

Strengths:

- Covers all major theoretical approaches
- Real Case Studies from range of Industries

Weaknesses:

- Contradictions between rival approaches
- We simply don't know what is best, where

© Ian Alexander 2006

32

Systematic Use of Scenarios

- Scenarios consist of Steps
- At each step, ask:
 - *What could go wrong here?*
 - *Is this step necessary?*
 - *Is this step in the right place?*
- This uses the Scenario's structure to guide the search for Exceptions and other issues

© Ian Alexander 2006

33

Scenario-Directed Search for Exceptions

At each step:

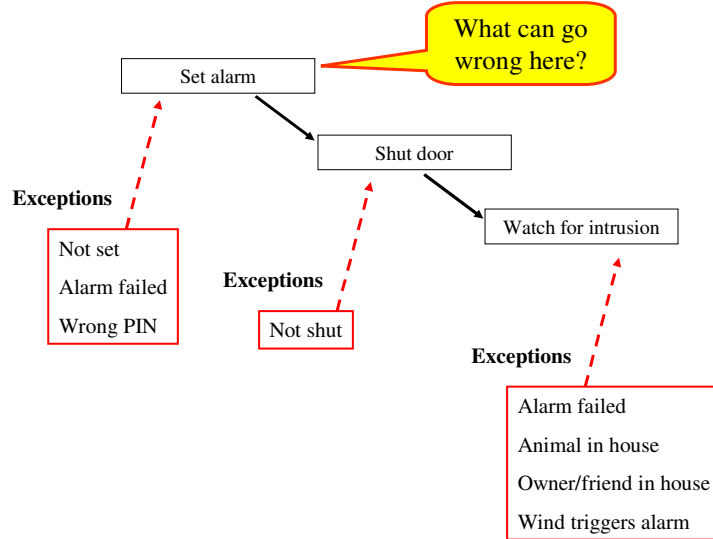
1. Look for what could go wrong (Exception Events)
 2. Create an Exception for each realistic* issue
 3. Create a Scenario to handle each Exception
 4. Repeat the search within the new Scenarios.
- The number of Exceptions usually far exceeds the number of 'Normal' Scenarios.
 - Requirements (and software code) to deal with Exceptions often form a large percentage of the project.

* This implies Prioritising (see below) – you can't do everything.
But, ignoring a possible Exception means accepting
the system will fail if/when that Exception occurs.

© Ian Alexander 2006

34

Eliciting Exceptions



© Ian Alexander 2006

35

Interaction-Directed Search for Exceptions

- Scenario-directed search is far better than just looking at a list of functions
- But knowledge of interaction types enables us to be much more systematic
- Eg “Set the alarm” involves
 - a human agent (not trained)
 - giving a command to
 - a machine
- That kind of interaction is liable to several **Classes of Exception**, eg:
 - human forgets
 - human makes a slip (finger presses wrong button etc)
 - human gets confused, gives the wrong command
 - machine fails
 - ...
- Can systematically (automatically, given a spreadsheet / database) ask
 - ‘Could the <named type of operator> forget to do this <named action>?’ etc,
 - for all steps of this kind
- Can be still more specific for a known domain (eg Air Traffic Control)

© Ian Alexander 2006

36

Requirements



© Ian Alexander 2006

37

Requirements Writing

- Quality, not perfection
 - no such thing as a perfect requirement
 - expect to draft, discuss, rewrite, review
 - Sketch, then improve
 - get the gist of the stakeholder's meaning in a deliberately rough sketch
 - find out, and write down, each requirement's justification
- ✦ *think out how you will test each requirement;
write down the result that will constitute passing the test*

© Ian Alexander 2006

38

Writing Functional Requirements

Role (<i>noun phrase</i>):	The call centre operator ...
Result / Action (<i>verb</i>):	... shall be able to view ...
Object (<i>noun phrase</i>):	... details of a protected household ...
Qualifier (<i>adverbial phrase</i>):	... within 2 seconds of issuing a query.

- This is a traditional style for functions.
- If you don't like "shall"s, use the *indicative*, e.g.
 '**The operator views the details ...**'
 and add an attribute to indicate priority, e.g.
 '**Essential**', '**Required in Phase 1**'

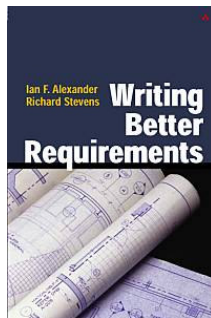
Can we
test that?

© Ian Alexander 2006

39



Structure of a Requirement



Ian Alexander & Richard Stevens
Writing Better Requirements
Addison-Wesley 2002

Key Concepts:

- Requirements are a communication between people with different backgrounds
- Requirements must tell a story clearly
- Requirement has many parts (not just text)
- Check, Review, Check some more

Strengths:

- Simple, practical industrial approach
- Ideal for use with RM tools such as DOORS

Weaknesses:

- Does not attempt to cover life-cycles, etc

© Ian Alexander 2006

40

Requirements as Tables

- Complicated (sets of) requirements are often much easier to define and to read in table form
- Can be mathematically precise (with functions in place of words)
- Can sometimes be translated straight into software.

Software too can be table-driven

Door sensor is triggered	Alarm is reset within time-out period	Log "normal entry"
	Otherwise	Send "suspected intrusion" to CC; Sound local bells
Non-door sensor is triggered		Send "suspected intrusion" to CC; Sound local bells

SensorType(S) == Door;
triggered(S)

© Ian Alexander 2006

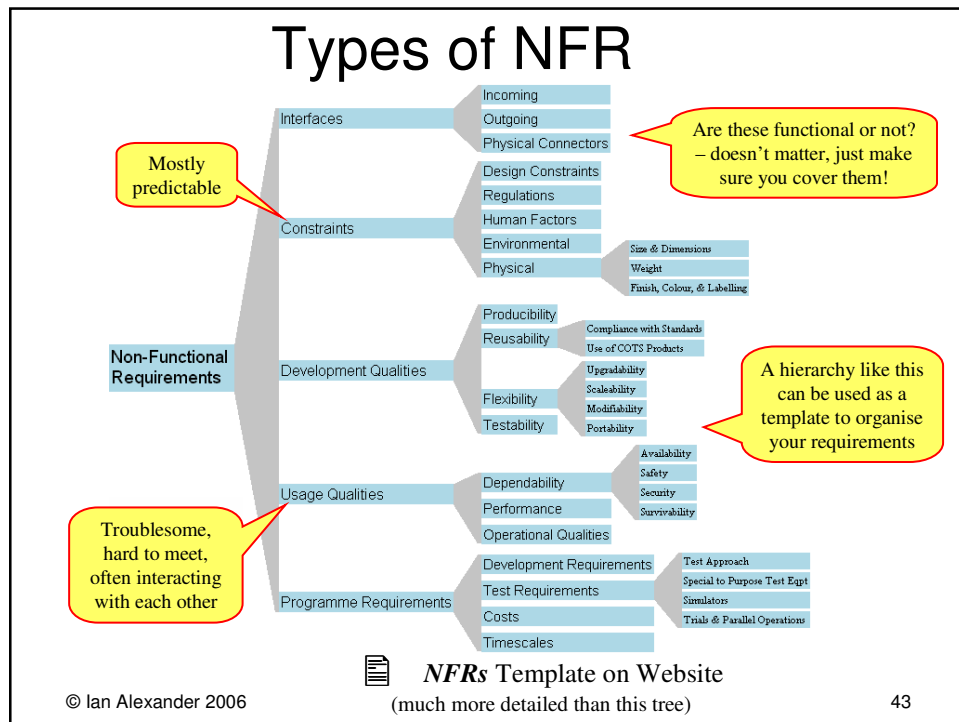
41

Non-Functional Requirements

- Requirements that constrain the desired system or development process in any way (other than by saying what the product is to do)
- Should be linked to specific functions
- Some NFRs are global
 - e.g. end-to-end performance targets
 - these are broken down into allocations to different subsystems

© Ian Alexander 2006

42



Effect of NFRs on System

- Reduce design freedom
- Increase complexity
- Increase cost
- *eg Availability 24hrs/day ...
... cuts the design choices and multiplies cost*
- Can make systems impossible to build
- ☞ *Don't allow any '-ilities' that are not absolutely necessary*
- ☞ *Handle changes introducing new '-ilities' with extreme care*

Example Constraints

ID	System Requirements for NADB Contractor	Acceptance Criterion	Justification	Verification Method	Priority	Review Status
N-9	5.6 Physical				---	---
N-29	5.6.1 Mechanical Construction RIA13 Clause 4. Construction G8621 Clause 4.1 Equipment Construction.	Passes successfully type and shock/vibration tests including visual inspection as defined in RIA13.	Mechanical operation environment of NADB.	Design Review Design Statement	Essential	---
N-30	5.6.2 Material Safety				---	---
N-1069	All electrical cables shall comply with RSE/STD/024 Part 6.	Documentary evidence backed by manufacturers data sheets to demonstrate compliance with the standard.	Fire hazard and toxic fume emissions.	Design Review Design Statement	Essential	Accepted
N-1070	All crimped terminations shall comply with E6487.	Sample wire pull tests to demonstrate compliance to the specification.	Harsh mechanical and electrical environment.	Design Review Design Statement	Essential	Accepted

© Ian Alexander 2006

45

Example '-ilities'

ID	System Requirements for NADB Contractor	Acceptance Criterion	Justification	Verification Method	Priority	Review Status
N-12	5.7.1 Reliability & Availability				---	---
N-1080	The contractor shall provide a reliability prediction, based on random component failure, to MIL-HDBK-217F, parts stress analysis, ground mobile, ambient environment at +55°C. The reliability so predicted for each NADB shall be at least 50,000 hrs MTBF of powered-up operation	MTBF calculation data sheets with a method statement and assumptions shall be submitted for the approval by the PE.	Reliability and availability of NADB (trains).	Design Review Design Statement	Essential	Accepted
N-14	5.7.2 Maintainability				---	---
N-1081	The NADB shall be a Line Replaceable Unit (LRU) which is a modular approach to design in order to permit rapid restoration of service by on-train replacement of the complete module in preference to in-situ repair. It shall not take more than 5 minutes to replace a faulty unit.	Able to replace the NADB in 5min.	Availability of train service.	Design Review Design Statement	Essential	Accepted
N-1084	The Mean Time To Repair (MTTR) of all Service Affecting Failure (SAF) shall be 2 hours, taken from the time of commencement of diagnostic checks to test and confirmation of fitness for service.	Able to diagnose ten separate faults within two hours per fault.	Ease of repair determines cost.	Bench Test Design Review Design Statement ATE	Essential	Accepted

© Ian Alexander 2006

46

Verifying NFRs

- ***NFRs often can't readily be verified by Test***
 - Safety is demonstrated by a safety case, leading to a **Certificate**
 - Reliability is calculated from equipment failure rates, ie by **Analysis**
 - Size, Weight, Finish and similar constraints can be checked by **Inspection**
 - Performance (given peak traffic, etc) can be predicted by **Simulation**

🔑 *Plan out how you'll verify your NFRs well in advance*



The *Non-Functional Requirements* Template suggests suitable approaches

© Ian Alexander 2006

47

Writing Non-Functional Requirements

- **Maintainability:**
'Supplier's hotline shall analyze 95% of reports within 2 work hours.'
- **Security:**
'Product shall safeguard against viruses that delete files.'
- **Usability:**
'Novice users shall be able to perform task Q in 15 minutes.'
- **Performance:**
'Product shall detect speed violation and take photo within 0.5 s.'

How robust are these requirements? Can you do better?

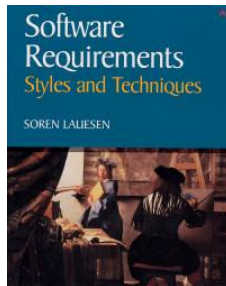
Examples from Lauesen

© Ian Alexander 2006

48



Requirement Styles



Søren Lauesen

*Software Requirements:
Styles and Techniques*

Addison-Wesley 2002

Key Concepts:

- different Styles for each sort of requirement
- e.g. Security, Maintainability, Performance

Strengths:

- practical experience e.g. of Shipyards
- detailed real examples
- elicitation techniques

Weaknesses:

- intentionally focuses on early stages (customer's point of view)
- semi-detached from mainstream thinking (maybe he's too pragmatic for a researcher!)

© Ian Alexander 2006

49

Justifications



© Ian Alexander 2006

50

Justifications

- by writing a brief text (eg a “Justification” attribute)
 - is simple, and easy to find
 - (it’s right beside the requirement text!)
 - but may be similar/identical for several requirements

OR

- by tracing to Goals, Business Requirements, etc
 - Goal Models etc may consist of many interlinked items
 - powerful, but takes time to follow (and some getting used to)

OR

- by modelling Rationale (and tracing to that)
 - Rationale may consist of many interlinked Assumptions (*see below*)
 - really heavy-duty, suitable for complicated arguments
 - but takes more time and effort to document

© Ian Alexander 2006

51

Justification as Text Attribute

ID	Requirement	Justification
331	Each order is traceable to the person who created it.	To enable queries or complaints to be handled; to suppress fraud by employees.

Use of Text allows you to supply several justifications if need be (eg 2 here)

But you may end up with a lot of duplication if some reasons apply to many requirements

© Ian Alexander 2006

52

Justification by Tracing to Sources

107 Each order is uniquely identifiable.

The reason (in this case, another requirement) is stated only once, but may justify many later decisions

satisfies

No text needs to be added to justify requirement 324: a link does the job

324 The identifier of a deleted order is never reused.

The price for this power is a requirements database (or at least, a hypertext)

© Ian Alexander 2006

53

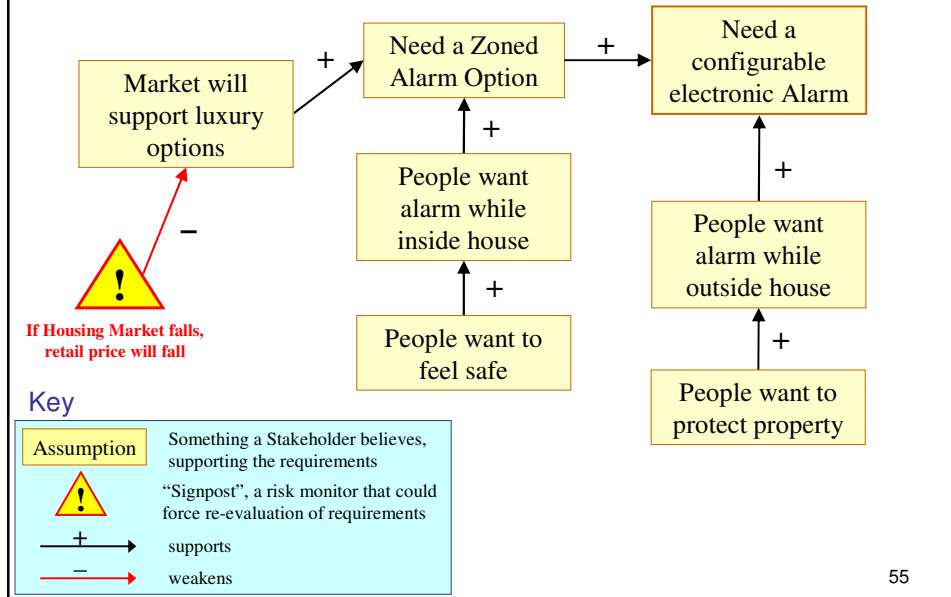
Assumptions



© Ian Alexander 2006

54

Analysing Assumptions



55

Agreed Priorities

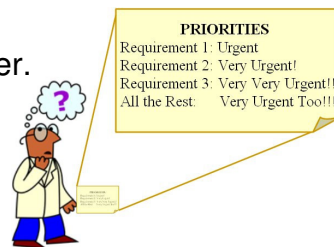


© Ian Alexander 2006

56

Prioritise

- You can't afford to do everything
- And it would take too long anyway
- If every requirement (from every stakeholder) is stamped "Mandatory"...
- ... how can the project / product manager choose?
 - "Randomly" is the wrong answer.
- So, prioritise.



© Ian Alexander 2006

57

Prioritised Requirements Example

ID	Goal Model and User Requirements	Performance	Review Status	Priority
UR-33	5.1.1.3.1.2 Detect intrusion The alarm detects a burglar in the protected household.			
UR-34	5.1.1.3.1.2.1 Notify intrusion			
UR-56	The alarm shall notify intrusion to the alarm center.	within 30 seconds of detection	Accepted	Essential

- Database tracks each item, allowing audit and tracing
- Whatever tool you're using, you need attributes/fields for Acceptance Criteria, Issues, Review Status and Priority

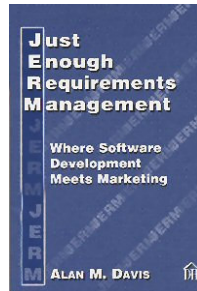
➡ Every requirement must have a record Identifier (like UR-34)

➡ Assign Priorities cautiously – only a few things can be Essential

© Ian Alexander 2006

58

Prioritisation



Alan Davis
Just Enough Requirements Management
Dorset House 2005

Key Concepts:

- Just enough, in time, is a whole lot better than really polished, too late for your product to get to market.

Strengths:

- Practical, fresh, full of anecdotes, intellectually sound. The first book that really talks about Triage. Combines requirements theory with business practice.

Weaknesses:

- Not the easiest place for a beginner to start.
- It talks about software but applies to all systems.

© Ian Alexander 2006

59

Triage

1. some requirements are so obviously essential that they do not need to be prioritised:
 - just do them.
2. some requirements are so obviously out of scope that they do not need to be prioritised:
 - just drop them.
3. In between is a third group of requirements worth taking time to prioritise:
 - rank them, vote on them, etc.

© Ian Alexander 2006

60

Risk-based Prioritisation

- Size of a Risk = Probability $\times$ Severity

it will occur
when it does
- Eg
 Severity (steering fails on your car while you're driving on the motorway) = *maximum*[†]
 $\therefore$ Acceptable Probability = *extremely low*^{*}
 $\therefore$ Priority (steering is reliable) = *Essential*

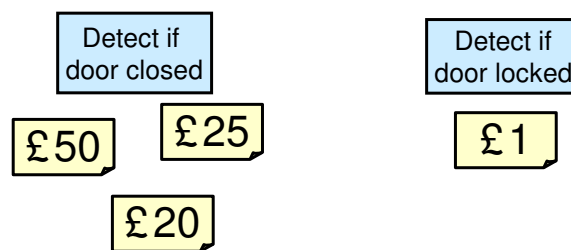
† *The effect on the business may be just as serious as the effects on the road.*

* *Goal: steering never fails*
Requirement: $p(\text{fails}) < 10^{-8}$ (per hour)
(but how would you prove that?)

© Ian Alexander 2006

61

Ranking



- Many possible ways to give requirements relative scores
- Giving each stakeholder “£100” (to write on post-it notes) to allocate among his/her favourite requirements is easy and fun

© Ian Alexander 2006

62

Your 7 Key Requirements

- Military projects use Key Requirements to identify their top priorities
- They are the ones you absolutely insist are delivered...
- ... or you won't pay up.
- You can only have 7 of them!
- What are the 7 Key needs for an aircraft carrier?
 - Let's try it...
- You can (of course) break down Key Requirements
 - eg using a Goal Model to identify sub-goals
 - The priorities of sub-goals can be derived from those of their parents/ancestors

Acceptance Criteria



Verifiability (1)

- Each requirement must be testable
 - define tests to check whether or not the requirement has been met
- Inventing tests for requirements is the best way to show you know how to test them
- Each requirement must have an Acceptance Criterion

☞ *Make it standard practice to have Acceptance Criteria and Verification Method columns beside your requirements*

☞ *Get your test people involved from the start*

© Ian Alexander 2006

65

Scenario-Based Testing

Scenario

- says what result is wanted
- may branch, have alternatives, allow parallel steps, ...
- defines preconditions, roles, systems, ...

Test Case

- says what result is wanted
- must consist of **definite and repeatable steps**
- defines **criteria** for accepting/rejecting
- provides **slots** for pass/fail, tester, date

© Ian Alexander 2006

66

Example: Scenario, Test

Scenario

- Detect flying insect (without sounding alarm)

Test

Generate Exception!

- Action:
 - Release flying bee (length 1 cm) into sensor test room
- Desired Response:
 - Detect flying insect
- Acceptance Criteria:
 - bee is detected within 15 seconds
 - alarm does not sound
- Result **Pass**
- Tester **J Smith**
- Date **11 Sept 2006**

© Ian Alexander 2006

67

Verifiability (2)

- “Verification Method” attribute
 - value is any combination of {Test|Inspection|Analysis|Simulation}
 - **Test**: any kind of physical test of the product
 - may need to break this down further eg Bench Test, On-Train Test, ...
 - **Inspection**: eyeball examination, ultrasound, ...
 - **Analysis**: any kind of calculation including mathematical proof
 - **Simulation**: computer modelling to demonstrate how the hard-to-test product will behave (Mars lander, nuclear plant, etc)

© Ian Alexander 2006

68

Formalising Verifiability (3)

- A “Verifiable?” flag can also be helpful
 - {True|False|—} or
 - {Certified|Not Certified|—} (if you prefer)
 - certifies that someone
(ID known to the database) has inspected the
requirement and believes it to be Verifiable

Conclusion



Can you 'Engineer' Requirements?

- Requirements are the foundation for engineering development of systems of all kinds, including software
- But requirements don't begin from a firm position
- Instead, requirements work creates that solid ground
- Not quite like any other branch of engineering

© Ian Alexander 2006

71

So is it just a black Art, then?

- Each situation *is* different
 - Sometimes security or safety are the prime concerns (even above functionality)
 - Sometimes the environment is full of constraints (legacy code, existing interfaces)
- *But* the simple, general techniques we've discussed usually help
 - stakeholder analysis, goals, scenarios, dictionary, acceptance criteria ...

© Ian Alexander 2006

72

Materials

- Information Model

Attached below

- Booklist

Attached below



- Glossary

see separate handout

- Templates

- Objectives
- Stakeholders
- Use Cases
- NFRs

free from my website

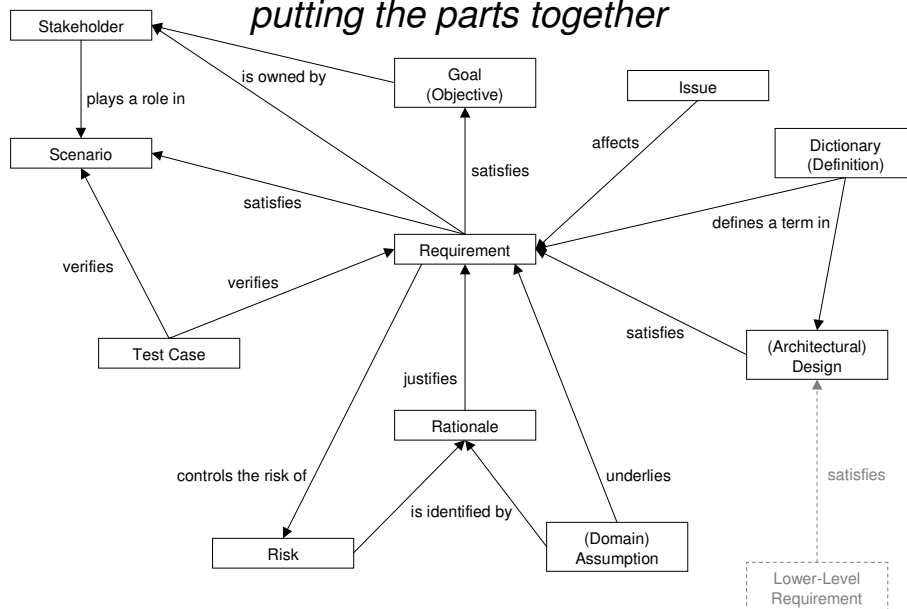
<http://www.scenarioplus.org.uk>

© Ian Alexander 2006

73

Information Model

putting the parts together



© Ian Alexander 2006

74

Booklist

Ian Alexander & Richard Stevens

Writing better Requirements 2002 – simple, practical writing guide

Ian Alexander & Neil Maiden

Scenarios, Stories, Use Cases 2004 – state-of-the-art on scenarios

Alistair Cockburn

Writing Effective Use Cases 2001 - invaluable on use cases

Ellen Gottesdiener

Requirements by Collaboration 2002 – lively, practical facilitator's approach

Michael Jackson

Software Requirements & Specifications 1995 - reflective; for ideas

Benjamin Kovitz

Practical Software Requirements 1999 - light, readable advice from a practitioner

Soren Lauesen

Software Requirements - Styles and Techniques 2002 - useful, detailed examples

Suzanne & James Robertson

Mastering the Requirements Process 1999 - simple detailed method (using events)

Alistair Sutcliffe

User-Centred Requirements Engineering 2002 – a detailed HCI perspective on requirements

Bill Wiley

Essential System Requirements 1999 - event-driven approach

Ralph Young

Effective Requirements Practices 2001 – practical manager's viewpoint

*Read reviews of these and other requirements books
at <http://www.scenarioplus.org.uk>*

© Ian Alexander 2006

75