Scenarios, Use Cases and Requirements in RESCUE

Centre HCI Design
25th March 2003
Learning Objectives

Six key learning objectives for participants

– **Understand the importance and benefits** of use cases and scenarios when establishing testable stakeholder requirements

– Be able to **write structured, concise use cases** in order to structure requirements and generate scenarios

– Be able to **walkthrough and facilitate walkthroughs of ART-SCENE scenarios** to discover and write testable stakeholder requirements

– Be able to **write requirements** in the style of the VOLERE requirement shell

– Be familiar with the purpose and nature of the VOLERE Quality Gateway

– Be able to **write requirements** that are able measurable, and hence testable
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00</td>
<td>RESCUE use case template</td>
</tr>
<tr>
<td>09.15</td>
<td>Writing and structuring normal course use cases with the RESCUE template, with exercise</td>
</tr>
<tr>
<td>10.15</td>
<td>Scenario generation from use case specification</td>
</tr>
<tr>
<td>10.30</td>
<td>Structure of a RESCUE scenario in ART-SCENE</td>
</tr>
<tr>
<td>10.40</td>
<td>Scenario Presenter: Features and walkthroughs</td>
</tr>
<tr>
<td>11.40</td>
<td>Practising a scenario walkthrough</td>
</tr>
<tr>
<td>12.30</td>
<td>Lunch</td>
</tr>
<tr>
<td>13.30</td>
<td>What happened with CORA-2 walkthroughs</td>
</tr>
<tr>
<td>13.45</td>
<td>VOLERE requirement shell, Requisite Pro and Quality Gateway</td>
</tr>
<tr>
<td>14.15</td>
<td>Writing measurable fit criteria for requirements</td>
</tr>
<tr>
<td>15.30</td>
<td>Linking requirements to use cases</td>
</tr>
<tr>
<td>16.00</td>
<td>ACRE: acquiring requirements</td>
</tr>
</tbody>
</table>
Part 1: Introduction and Inputs to Use Cases and Scenarios Stream
The Use Cases Stream

First synchronisation stage
- Gather data on human processes
- Model human activity
- Determine system boundaries
- i* context model
- Descriptions of human activities
- i* context model

Second synchronisation stage
- Determining system dependencies, goals, and rationale
- Creative design workshops
- i* SD and SR models

Third synchronisation stage
- Refine system dependencies, goals, and rationale
- Extended use cases
- Refined i* SD and SR models

Fourth synchronisation stage
- Use cases
- Defined use case specifications
- Use case model and summaries
- System-level requirements and associated use cases
- Use cases model and summaries
- Requirements
- Define system-level requirements
- Use case descriptions
- Define and document requirements
- Use case specifications
- Define and document requirements
- Use case specifications and associated requirements
- Walkthrough scenarios
- Define and document requirements
- Scenarios associated with requirements
- Impact analysis
- Refine and change requirements
Why Use Cases and Scenarios?

Why scenario-driven walkthroughs with use cases?

- Scenarios are an effective and cost-effective technique for acquiring testable stakeholder requirements
- Use cases offer an effective structure for communicating requirements to stakeholders
- Use cases provide possible hooks into the Rational Unified Process (RUP) already used in Eurocontrol
- Use case models can be automatically or semi-automatically derived from i* models
- Use case descriptions enable specifications from which we generate scenarios in CREWS-SAVRE
Part 2: RESCUE Process Template
Use Case Template

RESCUE guides use case description
- Through a comprehensive template that is both complete and simple-to-use
- Complete one template for each use case in the use case diagram, thus ensuring simple traceability
- Living document that is evolved as the device to structure all requirements in the requirements document

Similar to best-practice elsewhere
- Alistair Cockburn's best practices in use cases
  - [http://alistair.cockburn.us/usecases/usecases.html](http://alistair.cockburn.us/usecases/usecases.html)
Common Use Case Attributes

Best-practice drawn together from diverse sources

- Use case name, number and requirement(s)
- The normal course to achieve the requirement(s)
- All extensions and alternative courses
- Models (activity and sequence diagrams) showing the concurrent actions and agent message passing
- Class diagrams linked to the use case
- Related background information, references and comments
- Change history for the use case
- Open and unresolved issues for the use case
- Glossary of terms

RESCUE use case template as MS-Word document
# RESCUE Use Case Template

<table>
<thead>
<tr>
<th><strong>Name of Use Case</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Case ID</strong></td>
<td>Unique ID for Use Case</td>
</tr>
<tr>
<td><strong>Author</strong></td>
<td>Name of author</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>Date Use Case was written</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Source of Use Case</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Actors involved in Use Case (from Use Case Model)</td>
</tr>
<tr>
<td><strong>Problem</strong></td>
<td>Description of current problem</td>
</tr>
<tr>
<td><strong>Precis</strong></td>
<td>Informal scenario description</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>Requirement that the use case <strong>DECOMPOSES</strong></td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Requirements that impose <strong>CONSTRAINTS</strong> on the required behaviour of the use case</td>
</tr>
<tr>
<td><strong>Added Value</strong></td>
<td>Benefits of Use Case above and beyond the initial scenario in the original system</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>Why is the Use Case needed?</td>
</tr>
<tr>
<td><strong>Triggering Event</strong></td>
<td>Event or events that can trigger the Use Case</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>Necessary conditions for the Use Case to occur</td>
</tr>
<tr>
<td><strong>Assumptions</strong></td>
<td>Explicit statement of any assumptions made in writing the Use Case</td>
</tr>
<tr>
<td><strong>Successful end states</strong></td>
<td>Successful outcomes of the Use Case</td>
</tr>
<tr>
<td><strong>Unsuccessful end states</strong></td>
<td>Unsuccessful outcomes of the Use Case</td>
</tr>
</tbody>
</table>

## Normal Course

1. **Event 1**
   - **[EN]** System requirements **ENABLES** Event 1
   - **[CO]** System requirements **CONSTRAINTS** Event 1

2. **Event 2**
   - **[EN]** System requirements **ENABLES** to Event 2
   - **[CO]** System requirements **CONSTRAINTS** Event 2

...  

## Variations

1. If **condition** then **variation statement** (related to Event 1)

...  

## Alternatives

1. If **condition** then **alternative course statement** (related to Event 1)

...
Writing the Use Case Precis

Informal use case descriptions

- Essential element of RESCUE’s first stage, providing input to the creativity workshops
- Describe each use case using simple attributes (ID, source, actors) and precis (one paragraph) describing how actors will use the future system to achieve their goals
- Precis should be describe wider system vision

Essential affordances of these descriptions

- Avoid over-commitment by stakeholders
- Flexible representation, simple to change
- Focus on envisioned behaviour at expense of details
- Encourage people to work with and manipulate these representations before agreeing details
# Where the Precis Goes

<table>
<thead>
<tr>
<th>Requirement (S)</th>
<th>Requirement statement: decomposes the use case</th>
<th>Decomposes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraints</td>
<td>Requirement statement: constrains the use case</td>
<td>Constrains</td>
</tr>
<tr>
<td>Added Value</td>
<td>Benefit of use case above and beyond the original scenario of the use case</td>
<td>Added Value</td>
</tr>
<tr>
<td>Justification</td>
<td>Why is the use case needed?</td>
<td>Justification</td>
</tr>
<tr>
<td>Triggering Event</td>
<td>Event or events that can trigger the use case</td>
<td>Triggering Event</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Necessary conditions for the use case to occur</td>
<td>Preconditions</td>
</tr>
<tr>
<td>Assumptions</td>
<td>Explicit statement of any assumptions made in writing the use case</td>
<td>Assumptions</td>
</tr>
<tr>
<td>Successful End States</td>
<td>Successful outcome(s) of the use case</td>
<td>Successful End States</td>
</tr>
<tr>
<td>Unsuccessful End States</td>
<td>Unsuccessful outcome(s) of the use case</td>
<td>Unsuccessful End States</td>
</tr>
</tbody>
</table>

## Normal Course

1. Event 1
   - [EN] System requirement enables Event 1
   - [CO] System requirement constrains Event 1

2. Event 2
   - [EN] System requirement enables to Event 2
   - [CO] System requirement constrains Event 2

...  

## Variations

1. If [condition] then [variation statement] (related to Event 1)

## Alternatives

1. If [condition] then [alternative course statement] (related to Event 1)
Example CORA-2 Use Case Precis

Manage Invalid Resolution(s)

Should a resolution become invalid due to e.g. the expiration of its 'minimum time for implementation' G, CORA 2 deletes it from the stored resolutions and re-orders the ranking of the remaining resolutions. Should a resolution become invalid during co-ordination, CORA 2 requests the Systems Co-ordinator to discontinue the co-ordination process.

Comments on the precis

– The precis is largely design-free, although there are a number of tacit assumptions made
– Too few use case “steps”, suggesting perhaps that this is not a complete use case
Example CORA-2 Use Case Precis

Display Best-ranked Resolutions

On request, CORA 2 displays a set of ranked resolutions. One of each type G is displayed (if available), i.e. Climb, Descend, Left Turn, Right Turn and Adjust Speed. In parallel, CORA 2 displays the associated 'conflict information' G (including the related 'context aircraft' G), the cost-value on which the ranking is based and an indication of the 'optimum time to implement' G the resolutions. This allows the Controller to be aware of possible solutions for a conflict, and to understand and evaluate them rapidly before making a decision, i.e. (i) chose one of the CORA 2 resolution proposals, (ii) modify it or (iii) create his/her own resolution.

Comments on the precis

– More description of use case “steps”
– Perhaps too little wider vision for the system behaviour
Use Case Description Exercise

Purpose

- To practice simple use case precis as a starting point for further, more structured use case description

Task

- For the *Countdown automated bus indicator system*, write a use case precis for one or more of possible use cases modelled in the previous exercise
- Explore how simply these precis can be changed to incorporate new or changing requirements and design ideas
Part 3: Writing and Structuring Normal Course Use Cases
Where the Normal Course Goes

<table>
<thead>
<tr>
<th>Name of Use Case</th>
<th>Use Case ID</th>
<th>Author</th>
<th>Date</th>
<th>Source</th>
<th>Actors</th>
<th>Problem statement (now)</th>
<th>Precis</th>
<th>Requirement s</th>
<th>Constraints</th>
<th>Added Val ues</th>
<th>Just iation</th>
<th>Triggering event</th>
<th>Precond itions</th>
<th>Assumptions</th>
<th>Successful end states</th>
<th>Unsuccessful end states</th>
<th>Normal Course</th>
</tr>
</thead>
</table>
|                  |             |        |      |        |        |                          |        | Requirement s | Requirements that impose CONSTRAINTS on the required behaviour in the use case | Benefit of Use Case above and beyond the original scenario in the original system | Why is the Use Case needed? | Event or events that can trigger the Use Case | Necessary conditions for the Use Case to occur | Explicit statement of any assumptions made in writing the Use Case | Successful outcomes of the Use Case | Unsuccessful outcomes of the Use Case | 1. Event 1
  - [EN] System requirement ENABLES Event 1
  - [CO] System requirement CONSTRAINTS Event 1

2. Event 2
  - [EN] System requirement ENABLES to Event 2
  - [CO] System requirement CONSTRAINTS Event 2

... |
| Variations | 1. If condition then [variation statement] related to Event 1 |
| Alternatives | 1. If condition then [alternative course statement] related to Event 1 |
|            |... |
Structured Authoring of Normal Course

RESCUE draws on several specific sources

- CREWS-SAVRE style guidelines to ensure completeness and correctness of the normal course
- CREWS-ECRITOIRE content guidelines, to ensure a well-written use case amenable to some form of computational analysis
- Temporal semantics from Allen (1985) for expressing concurrent actions
- A CORA-2 lexicon specialised to the ATM domain for expressing actions, agents and objects in a normal course

Simple-to-learn, simple-to-use

- Training and practice sufficient to compile the skills needed to author use cases
CREWS-SAVRE Style Guidelines

Simple style guidelines to consider

S1 Each action is undertaken by one agent
S2 Write normal course as discrete actions on new lines
S3 Avoid sentences with more than 2 clauses
S4 Use consistent agent, object and action names
S5 Avoid synonyms and homonyms (he, she, them, it)
S6 Use present tense and active voice
S7 Avoid use of negations, adverbs and modal verbs
S8 Bring out and make explicit the sequences, conditions and iterations in use case descriptions
S9 Separate out the contextual information
S10 Make clear all calls to other use cases
S11 Use graphics, storyboards and video clips to support actions that cannot be expressed solely with text
CREWS-ECRITOIRE Content Guidelines

Actions undertaken by different types of agents

- `<actor>....<action>`, e.g. the controller makes a decision

- `<actor>....<action>....<clause>`, e.g. the controller makes a decision to allocate ambulance to an incident.

- `<time or sequence factor>.... <actor> ....<action> ....<constraints>`, e.g. after appraising all relevant information, the controller makes a decision to allocate an ambulance to an incident.

- `<actor>....<action>.....<noun>.......<state>......<state>`, e.g. the controller sends the incident report from his desk to the collection point
Describing Concurrent Actions

Concurrent actions, from Allen (1985)

- **Strict sequence (A then B):** ev(endA)<ev(startB);
- **Part sequence (A meanwhile B):** (ev(startA)<ev(startB)) AND ((ev(endA)>(ev(startB));
- **Inclusion (A includes B):** (ev(startA)<ev(startB)) AND ((ev(endA)>(ev(endB));
- **Concurrent (A and B):** No rules about ordering of events;
- **Alternative (A or B):** (ev(startA) occurs) OR (ev(startB) occurs);
- **Parallel and equal (A parallel B):** (ev(startA)=ev(startB)) AND ((ev(endA)=(ev(endB));
- **Equal-start (A starts-with B):** (ev(startA)=ev(startB)) AND ((ev(endA)not=(ev(endB));
- **Equal-end (A ends-with B):** (ev(endA)=ev(endB)) AND ((ev(startA)not=(ev(startB)).
Communication-Type Actions

Interact with

Input

Modify flight data  Transfer  Assume  Change settings

Read

Monitor  Review  Detect  Validate  Scan

Enter flight data

ATC setting  Aircraft setting

Set frequency  Set QNH  Set transponder

Identify
Cognitive-Type Actions

Cognitive resolution action

- Project air situation
- Select
- Prioritise
- Form mental picture
- Calculate

Non-resolution cognitive action

- Check estimates
- Check optimisations
- Check LOAs
Communication-Type Actions

Agents communicate

Electronic communication

Talk

Listen

Gesture

Instruct

Request

Provide info

Coordinate

Respond

Electronic

Signal

Point

Change frequency

Issue clearance

Standby

Acknowledge

Confirm

Approve

Reject

Climb

Descend

Route

Heading

Read back

See next page
Communication-Type Actions

- Electronic communication
  - ATCO
    - Co-ordinate
      - Propose
      - Accept
      - Reject
      - Counter Propose
    - Transfer
      - Propose
      - Accept
  - Aircraft
    - Correlate
    - Squawk
    - Interrogate
      - Ident
      - Code
Physical-Type Actions

Physical action

Aircraft action
- Climb
  - Fast
  - Slow
- Descend
  - Fast
  - Slow
- Speed change
- Routing
- Turn

Human action
- Press PTT button
- Use mouse
- Standard rate
- Non-standard rate
Objects

Radar
  - PSR
    - Long range
    - Short range
  - SSR
    - ‘Standard’
    - Mono-pulse
  - Mode-S
    - Controller Working Position
      - PVD Screen
      - Mouse
      - Comms panel
        - Phone
        - RT
        - Headset
Objects

Transponder-equipped aircraft

Civil

- VFR
  - Light aircraft
  - Balloon
  - Helicopter

- IFR
  - Air transport
  - Business jet
  - Civil transport
  - Cargo

Military on a mission

- High performance
- ‘Standard’ performance
Actors

- Agent
  - ATCO
    - Aircraft
      - Planning
      - Tactical
      - Previous sector
      - Next sector
Inputs to Writing Use Case Descriptions

Human activity models
- Descriptions of current human behaviour that constrain the possible behaviours and sequences of behaviour
- Cognitive task models that prescribe required human behaviour with the new system

Innovative design ideas
- Important design features and design constraints that influence actor behaviour

System goal models
- Actor dependencies from $i^*$ SD models
- Goals, tasks and resources from $i^*$ SR models

Use cases and requirements
- Use case precis, use case-level requirements
Supermarket Checkout Use Case Example

Normal course

1. The operator swipes the customer's club card.
2. The system reads customer information from club card.
3. REPEAT while more products to be purchased.
   3.1 The operator swipes the product using the barcode reader.
   3.2 The system displays the item name and price.
   3.3 The system records purchase details from the product barcode.
4. The operator presses the 'transaction total' button.
5. The system displays the total amount due for the purchase.
6. The customer pays the operator.
7. The operator returns change to the customer.
8. The operator presses the 'transaction paid' button.
9. The system records the entire transaction.
10. The system updates the customer's club card details with the purchase information.
11. The system prints a receipt with all purchase details on the receipt.
CORA-2 Use Case Example

Consult Resolution use case

1. The CONTROLLER requests the available resolutions before the display of best-ranked reminder.
2. The CORA 2 SYSTEM displays the resolutions.
3. IN PARALLEL the CORA 2 SYSTEM displays information on validity time frame for each resolution.
4. IN PARALLEL the CORA 2 SYSTEM displays information on optimum time to act for each resolution.
5. IN PARALLEL the CORA 2 SYSTEM displays the associated conflict information.
6. IN PARALLEL the CORA 2 SYSTEM highlights the context a/c in relation to the conflict.
7. The CONTROLLER analyses the conflict information.
8. The CONTROLLER forms a mental picture of what resolution s (he) thinks is appropriate for the given conflict.
9. The CONTROLLER deduces his/ own possible resolution.
10. The CONTROLLER reviews the CORA 2 provided resolutions.
11. The CONTROLLER selects a specific resolution.
12. The CORA 2 SYSTEM displays the effect of the accessed resolution on the conflict.
13. IN PARALLEL the CORA 2 SYSTEM displays the contextual traffic affecting the resolution.
14. The CONTROLLER compares the CORA 2 displayed resolution with his/ her deduced resolution.
15. The CONTROLLER accepts the proposed resolution.
Use Case Description Exercise

Purpose

– To practice style and content guidelines for writing use case normal course descriptions

Task

– For the *Countdown automated bus indicator system*, write some short(-ish) use case descriptions for one or more of the use cases for which you wrote a precis in the previous exercise
– Demonstrate the use of the style and content guidelines
Part 4: Structuring and Writing Use Case Variations and Alternative Courses
Variations and Alternative Courses

What are variations?
- Different normal course actions, expressed as sequences, that also lead to the success end state for the use case
- Needs domain expertise and judgement to distinguish normal and variation actions
- Represent in separate section to normal course actions

What are alternative courses?
- Actions, and their pre-conditions, that lead to an unsuccessful end state for the use case
- Essential to ensure requirements completeness
- Automatically generated in the CREWS-SAVRE tool
| Name of Use Case | Use Case ID | Unique ID for Use Case | Author | Name of author | Date | Date Use Case was written | Source | Source of Use Case | Actors | Actors involved in Use Case (from Use Case Model) | Problem Statement (now) | Description of current problem | Precis | Informal scenario description | Requirements | Requirement(s) that the use case DECOMPOSES | Constraints | Requirements that impose CONSTRAINTS on the required behaviour in the use case | Added Value | Benefit of Use Case above and beyond the original scenario from the original system | Justification | Why is the Use Case needed? | Triggering Event | Event or events that can trigger the Use Case | Preconditions | Necessary conditions for the Use Case to occur | Assumptions | Explicit statement of any assumptions made in writing the Use Case | Successful end states | Successful outcome(s) of the Use Case | Unsuccessful end states | Unsuccessful outcome(s) of the Use Case | Normal Course | 1. Event 1 | [EN] System requirement ENABLES Event 1 | [CO] System requirement CONSTRAINTS Event 1 | 2. Event 2 | [EN] System requirement ENABLES to Event 2 | [CO] System requirement CONSTRAINTS Event 2 | ... | ... | Variations | 1. If [condition] then [variation statement] (related to Event 1) | ... | ... | Alternatives | 1. If [condition] then [alternative course statement] (related to Event 1) | ... | ... |
How to Discover and Write Variations

Discover variations using simple questions
- Are there other means of achieving the use case goal?
- Can these action(s) be undertaken by another agent?
- Can these action(s) be undertaken in a radically-different sequence?

Write each variation as a mini-normal course
- Action number: Normal course action(s) which the variation defines different behaviour to
- Precondition: WHEN-statement defining the precondition for the variation
- Action statements: Sequence of numbered action statements that describe how the actors shall behave when the precondition is true
Supermarket Checkout Variations Example

Consider portion of the use case

6. The customer pays the operator.
7. The operator returns change to the customer.
8. The operator then presses the 'transaction paid' button.

Some obvious variations on payment methods

6-8. IF the customer pays with credit card
6. The customer gives the credit card to the operator.
7. The operator swipes the credit card using the barcode reader.
8. The system reads the credit card number from the card.
9. The system validates the credit card number.
10. The system prints the credit payment slip.
11. The operator takes the credit payment slip from the system.
12. The customer signs the credit payment slip.
13. The operator validates the customer signature.
14. The operator presses the 'transaction paid' button.
CORA-2 Use Case Variations Example

From Consult Resolution use case

1.3. If the CONTROLLER doesn't consult the available before the display of best ranked reminder. Then

1.3.1. The CORA 2 SYSTEM generates a reminder to act on the best one resolution.

1.3.2. The CORA 2 SYSTEM displays the reminder.

1.3.3. The CONTROLLER detects the reminder.

1.3.4. DO USE Consult Resolution (s)
How to Write Alternative Courses

An alternative course follows a regular structure

– **Action number**: Normal course action(s) to which the alternative course pertains
– **Precondition**: IF-statement defining the precondition for the alternative course
– **Action statements**: Sequence of (unnumbered) action statements that describe how the actors shall behave when the precondition is true

Thus each alternative course is independent

– Independent of all other alternative courses
– Do not attempt flow charts and UML sequence and activity diagrams - *it will become too complicated*
– Alternative courses ensure complete coverage and provide a simple partial rationale for requirements
Discovering Alternative Courses

Guidelines to predict relevant alternative courses:

- Events (*not happening, too frequent, too infrequent, wrong order*)
- Actions (*insufficient information, not completing*)
- Cognitive exceptions (*slips, mistakes, lack of knowledge/skill*)
- Other human exceptions (*age, size, gender, disabilities*)
- Machine exceptions (*power failures, breakdowns, blockages*)
- Human-machine exceptions (*misinterpret interface*)
- Machine-machine exceptions (*communication failure, scrambled messages*)
- Environmental exceptions (*light, heat, humidity, noise*)
Some Supermarket Checkout Alternatives

Alternative courses

1. **IF the customer does not have a club card**, do not do action.
2. **IF the customer does not have a club card**, do not do action.
3.1. **IF the item does not have a bar code**. The operator places the item on the scale. The operator enters a 4-digit number which is a unique identifier for the fruit or vegetable. The operator presses the button. The system calculates the price.
3.1. **IF the bar code reader cannot read the product code**. The operator enters the code using the keyboard.
3.1. **IF one product is bought several times**. The operator swipes the bar code once. The operator enters the number of products to be purchased. The system multiplies the price by the number of products.
3.1. **IF there is a product code error**. The operator cancels the last bar code swiping operation. The system removes the last product from the current transaction.
3.1. **IF the product has a special price reduction**. The operator enters the special reduction price of the product.
3.3. **IF the item does not have a bar code**. The operator enters the bar code number of the product using the keyboard.
Use Case Description Exercise

Purpose

– To practice discovering and describing variations and alternative courses for use cases

Task

– For the *Countdown automated bus indicator system*, write possible variations and alternative courses for the earlier use cases
– Demonstrate again the use of relevant style and content guidelines
Part 5: Scenario Generation From Use Case Specifications
Generating CREWS-SAVRE Scenarios

Two-stage process

Develop use case specification
  - Manual process of validating use case description and adding parameters to it to enable scenario generation

Generate scenarios from use case specification
  - Enter the specification into CREWS-SAVRE
  - Algorithm generates one or more scenarios
  - Scenarios are loaded to ExcelPresenter and Scenario Presenters for use
Use Case Specification
MS-Word template completed by City staff
CREWS-SAVRE Domain Modeller
CREWS-SAVRE Exception Classes

Predict useful alternative courses
- “a state or event that is necessary but not sufficient for the occurrence of non-normative behaviour in the system or its environment”

Synthesise existing taxonomies

Data base of >100 exception classes
Generating Normal Course

Scenario is permissible sequence of events

- Start and end actions ordered according to temporal semantics: **Strict sequence** (A then B), **Part sequence** (A meanwhile B), **Inclusion** (A includes B), **Concurrent** (A and B), **Alternative** (A or B), **Parallel and equal** (A parallel B), **Equal-start** (A starts-with B), **Equal-end** (A ends-with B)

- Simple example

```
Scenario1

Event S1
Action 1
Event S3
Event E1
Action 3
Event E3
Event E1

Scenario2

Event S1
Event S3
Event E3
Event E1
```
Generating Alternative Courses

Uses simple type model
- Action and agent types

- User-imposed constraints on alternative course generation, from use case specification parameters
ATM Exception Classes

System data exception

- EC1.1: Incomplete or interrupted data
- EC1.2: Inaccurate or incorrect data
- EC1.3: No data
- EC1.4: Radar data and pilot reported data inconsistent
  - EC1.4.1: Location
  - EC1.4.2: Level
  - EC1.4.3: Call sign

- EC1.2.1: Incorrect data on flight strip
- EC1.2.2: Incorrect wind data
- EC1.2.3: 2 Aircraft have same call sign
- EC1.2.4: Flight plan not updated
  - EC1.2.4.1: Level
  - EC1.2.4.2: Destination
  - EC1.2.4.3: Aircraft type
  - EC1.2.4.4: Route
  - EC1.2.4.5: Speed
ATM Exception Classes

Non-standard aircraft performance

- Unexpectedly high performance
  - EC2.1
- Unexpectedly low performance
  - EC2.2
- High performance military aircraft
  - EC2.3

Human-human communication exception

- Pilot has poor knowledge of English
  - EC4.1
- Controller not confident in communication partner
  - EC4.2
- Misinterpretation of communication
  - EC4.3
- Gesture not seen
  - EC4.4
- Gesture misunderstood
  - EC4.5
- Comment not heard
  - EC4.6
Controller cognitive exception

EC3

EC3.1 Conflict 'incorrectly' prioritised
EC3.2 Conflict 'incorrectly' classified
EC3.3 Risk 'incorrectly' calculated
EC3.4 Cognitive mistake
EC3.5 Lack of confidence in received data
EC3.6 Lack of confidence in own decision making
EC3.7 'Incorrect' plan selected
EC3.8 Task 'Incorrectly' prioritised
EC3.9 Lack of confidence in object or agent
EC3.10 Don't transfer aircraft to new sector

EC3.4.1 Mental block
EC3.4.2 Cognitively don't 'hear'
EC3.4.3 'Forget' to carry out conflict resolution plan
EC3.4.4 Don't spot conflict
EC3.4.5 Fail to check goal against plan
EC3.4.6 Fail to check goal against new data
EC3.4.7 Fail to carry out 'full scan' of screen

EC3.8 EC3.9

EC3.8

EC3.9
ATM Exception Classes

Radar exception

- EC5.1: One or more input radar fails
  - EC5.2.1: Transponder not selected
  - EC5.2.2: Transponder box failure
  - EC5.2.1.1: Military procedure
  - EC5.2.1.2: Pilot error
- EC5.2: Aircraft doesn’t squawk
- EC5.3: ‘Garbled’ data
- EC5.4: Poor radar performance
  - EC5.4.1: Radar ‘hole’
  - EC5.4.2: Radar provides inaccurate data
  - EC5.4.2.1: Incorrect speed
  - EC5.4.2.2: Incorrect level
  - EC5.4.2.3: Incorrect code
ATM Exception Classes

HMI presentation exception

EC6

- Incorrect tool selection
- Incorrect field selection
- Incorrect setting
- Window hidden
- Misinterpret another controller’s Windows presentation preferences
- Over-Reliance on CORA-2
- Congested screen
- Screen fails
- Screen freezes
- Mouse inoperative

Incorrect filter setting
Incorrect frequency setting
Incorrect range setting
Incorrect label setting
Incorrect speed vector setting
ATM Exception Classes

Pilot behaviour exception

EC7

EC7.1 Pilot not listening to messages
- Pilot misinterprets the instruction (EC7.2.1)

EC7.2 Pilot doesn't comply with the instruction
- Pilot unwilling to comply with the instruction (EC7.2.2)
- Pilot unable to comply with the instruction (EC7.2.3)

EC7.3 Pilot complies with plan, but not in the required time frame

EC7.4 Pilot using incorrect settings
- Incorrect frequency (EC7.4.1)
- Incorrect altimeter setting (EC7.4.2)
- Incorrect rate of climb (EC7.4.3)
- Incorrect speed (EC7.4.4)
- Incorrect SSR code (EC7.4.5)
- Incorrect heading (EC7.4.6)
ATM Exception Classes

Weather exception

- Cloud exception
  - Jet stream
    - EC8.2.1
  - Wind shear
    - EC8.2.2
- Wind exception
- Thunderstorm exception
- Temperature exception
  - Extreme heat
    - EC8.4.1
  - Extreme cold
    - EC8.4.2
- Turbulence
- Fog
- Icing
Airspace exception

- Heavy sector load
- Military airspace
- Uncontrolled airspace
- Special use airspace

Control room environment exception

- Noise
- Controller cannot see screen
- ‘Information overload’

- Light on screen
- Physical obstruction
ATM Exception Classes

Radio exception

- Congested frequency
- Radio erroneously set to transmit only
- Radio unavailable
- 'Clipped' message
- Interrupted message
- Interference from other radio sources
- Phone unavailable
- Radio failure
- Channel blocked

Special case

- System failure
- Headset failure
- Microphone failure
- Hijack
- Radio comm failure
- Emergency

Incorrect frequency & filter setting
Data input error causes Inaccurate mental picture causes Non-standard aircraft performance causes System data exception causes Pilot behaviour exception causes Airspace exception causes Controller cognitive exception causes Control room environment exception causes Human-human communication exception causes Radar exception causes Radio exception causes HMI presentation exception causes Inaccurate mental picture causes Controller inexperience
Inputs to Use Case Specification

Type model
- Define types of agents and objects in the use case

Parameterise alternative course generation
- Define classes/sub-classes of generic exceptions
- Define classes/sub-classes of air traffic control exceptions

Set up scenario generation templates
- Define templates for different stakeholder groups
- Define templates for different types of scenario walkthroughs - for different purposes
Part 6: Structure of a RESCUE Scenario in ART-SCENE
Common Scenario Structure

Standard four-part scenario structure
– Reflects structure of underlying scenario database

- Scenario edit and walkthrough functions
- Scenario normal course event sequence
- Explore across
- Scenario alternative courses, per normal course event
- Candidate generic requirements, per alternative course

Walk through
– Interactive software tools needed for walkthroughs
  • ExcelPresenter (MS-Excel) or Scenario Presenter (web-based)
Part 6: RESCUE Scenarios in the web-based Scenario Presenter tool
How Scenario Presenter Works

Scenario Presenter
– Each scenario is stored on a remote server and accessed using browser technologies
– Application supports IE, built with MS-InterDev

Advantages
– Integrated scenario walkthroughs if network access
– Scenario, requirement and comment sharing
– Better user interface and more walkthrough features

Disadvantages
– Reliability depends on internet and server reliability
– Each interaction requires a browser page refresh

Use for future project walkthrough processes
– Revised ExcelPresenter version remains as back-up
Scenario Structure During Walkthrough

Scenario normal course event sequence per normal course event
Shortened Normal Course Scenarios

Feature to show normal course start events only
Shortened Scenario Alternative Courses

<table>
<thead>
<tr>
<th>Main Events</th>
<th>Alternative Course Scenario 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Communication</td>
<td>1. What if the controller cannot see the screen?</td>
</tr>
<tr>
<td>2. Communication</td>
<td>2. What if the controller does not transfer the device to a new window?</td>
</tr>
<tr>
<td>3. Communication</td>
<td>3. What is an error exception?</td>
</tr>
<tr>
<td>4. Communication</td>
<td>4. What if the controller is hidden?</td>
</tr>
</tbody>
</table>

Feature to show selected alternative courses only
Adding and Viewing Requirements

List of requirements that have been added

Feature to add new requirement with key features
Viewing Requirements Within Scenario

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Action Type</th>
<th>Description (Natural Language)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Communication</td>
<td>The PC transfers the conflict responsibility to the TC</td>
</tr>
<tr>
<td>2</td>
<td>Communication</td>
<td>The PC transfers the conflict responsibility to the TC</td>
</tr>
<tr>
<td>3</td>
<td>Communication</td>
<td>The CORA 2 SYSTEM displays a conflict transfer message to the TC</td>
</tr>
<tr>
<td>FR3</td>
<td>Communication</td>
<td>It is essential to update the TC at every stage in the conflict resolution process. Neil Maiden</td>
</tr>
<tr>
<td>PR1</td>
<td>Communication</td>
<td>Normal performance characteristics are desirable for this event. Neil Maiden</td>
</tr>
<tr>
<td>PR2</td>
<td>Communication</td>
<td>The CORA 2 SYSTEM displays a conflict transfer message to the TC</td>
</tr>
<tr>
<td>PR3</td>
<td>Communication</td>
<td>The CORA 2 SYSTEM displays a conflict transfer message to the TC</td>
</tr>
<tr>
<td>PR4</td>
<td>Communication</td>
<td>The CORA 2 SYSTEM displays a conflict transfer message to the TC</td>
</tr>
<tr>
<td>PR5</td>
<td>Communication</td>
<td>The CORA 2 SYSTEM displays a conflict transfer message to the TC</td>
</tr>
</tbody>
</table>

View scenarios integrated within the scenario format, similar to use case structuring of requirements.
Editing Scenario Normal Course Events

Feature to edit normal course event/action
Adding Comments About Scenario

Feature to add comments to a scenario event during a walkthrough
Part 7: User Levels in the Scenario
Presenter
Scenario Presenter User Levels

System administrator controls access

- City University manages tool, in conjunction with local administrator at Eurocontrol

Each designated user has access at 1 of 4 levels

1. **System administrator**: full control administered by one person
2. **Scenario facilitator**: full access rights over one or more scenarios, and to collate comments from level-3 users about a scenario
3. **Scenario user**: read-only access rights over one or more scenarios, plus write-access to add comments and requirements related to a scenario
4. **Guest access**: read-only access to all features of one or more scenarios
### ART-SCENE Scenario Presenter

#### User List

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Username</th>
<th>Password</th>
<th>Digits</th>
<th>User Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>Smith</td>
<td>Alex</td>
<td>ABC</td>
<td>123</td>
<td>5</td>
</tr>
<tr>
<td>Sarah</td>
<td>Davis</td>
<td>Sarah</td>
<td>DEFG</td>
<td>456</td>
<td>2</td>
</tr>
<tr>
<td>John</td>
<td>Brown</td>
<td>John</td>
<td>HJKL</td>
<td>789</td>
<td>3</td>
</tr>
<tr>
<td>Michael</td>
<td>Johnson</td>
<td>Michael</td>
<td>MNOP</td>
<td>012</td>
<td>1</td>
</tr>
</tbody>
</table>

#### System Status Logon

- Admin
- Guest
- Manager
- Happy
- Role
- Todo
- Admin
- Hong

[Link](http://www.example.com/login.aspx)
Part 8: RESCUE Scenario Walkthrough
Environment
Scenario Walkthrough Environment

Prepare the walkthrough environment carefully
  – Provide relevant use case-driven requirements documents to participants

Clear roles of the facilitators
  – *Facilitator* leads the walkthrough, communicating with the participants and encouraging their involvement
  – *Scribe* records all results from the walkthrough, including comments, new or changed requirements, and scenario changes
  – *Participants* provide stakeholder input to the walkthrough process, to ensure requirement completeness and correctness
  – These roles and the walkthrough rules are communicated to all participants beforehand
Planning Scenario Walkthrough

Plan for half-day walkthrough
- 15-minute introduction, 150-minute walkthrough session, 15-minute wrap-up session

Which participants to invite
- One facilitator and one scribe are essential
- Representatives of all stakeholders/actors who undertake actions in the scenario
- Other key domain and technology experts

Create the right environment
- Walkthroughs should be creative and innovative, so provide a relaxing environment which does not inhibit involvement
- Learn from the creativity workshops
Before Scenario Walkthrough

All participants prepare for walkthrough
- To familiarise themselves with the scenario
- To edit the scenario so that it is agreed by all participants

Each participant reviews the scenario individually
- Recognise possible new requirements
- Send comments on scenario to facilitator
- Propose candidate changes to scenario to facilitator

Facilitator changes scenario in light of comments
- Consider all candidate comments and make agreed changes to the scenario
- Publicise scenario and changes to it to all participants
Scenario Walkthrough Environment

Physical layout of room is important
– Advise use a structure based on JAD/RAD workshops

1. Is event relevant?
2. Is event handled?
3. Add requirement
Scenario Walkthrough Process

At start of scenario walkthrough
- Facilitator introduces scenario and originating use case
- Facilitator summarises participant comments and earlier changes made to the scenario
- Participants agree and make final changes to scenario

During scenario walkthrough
- Walk through normal course to discover requirements for each event in turn
- Walk through each alternative course for each normal course event in turn, to discover new requirements

At end of scenario walkthrough
- Consider all requirements discovered during the walkthrough, to add, edit or delete requirements
Part 9: Practising Scenario
Walkthroughs with Scenario Presenter
Practising Scenario Walkthroughs

Purpose
– To practice a scenario walkthrough with ART-SCENE’s Scenario Presenter

Task
– Consider one CORA-2 scenario. Act as participants in a walkthrough in 2 stages. Firstly, consider the scenario and make comments (and proposed changes) to it. Secondly, be facilitated in the scenario walkthrough to discover possible requirements for it
Part 10: Lessons Learned from Scenario Walkthroughs with CORA-2
Results: The Basics

Successful walkthroughs

- Half-day walkthroughs of 10 scenarios with key stakeholders over 3-week period from Nov-Dec 2001
  - Total 254 normal course events, 3000+ alternative courses
- Effective process for discovering new requirements
  - Led to 189 new requirements on top of original 249, kept 134
  - New requirements tended to more fine-grained and complete
- Follow-on questionnaires to all stakeholders

<table>
<thead>
<tr>
<th>Av</th>
<th>Low</th>
<th>High</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>5</td>
<td>6</td>
<td>Some alternatives difficult to understand</td>
</tr>
<tr>
<td>5.2</td>
<td>4</td>
<td>6</td>
<td>Exhaustive elicitation process</td>
</tr>
<tr>
<td>4.6</td>
<td>2</td>
<td>7</td>
<td>Very impressed</td>
</tr>
</tbody>
</table>

Problems with alternatives
Results: Follow-Up Interviews

CREWS-SAVRE Strengths

- Walkthrough requirements tended to be more complete, stable and testable, with rationale
- Walkthrough process acquired both coarse- and fine-grain requirements
- Scenarios provided common ground for discussion
- Scenarios imposed the use of the project glossary

CREWS-SAVRE Weaknesses

- Skip events when requirements already known
- Stakeholders disagreed with some high-level design decisions taken into account when writing scenarios
- Some requirements were linked to multiple events in multiple scenarios
- Usability problems - scrolling, cell width, navigation
Results: Cost-Effectiveness

Were walkthroughs cost-effective?

- CREWS-SAVRE walkthroughs lead to the generation of more requirements in the same time period than other requirements acquisition techniques

<table>
<thead>
<tr>
<th></th>
<th>Before walkthrough</th>
<th>During walkthrough</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>10 months</td>
<td>3 weeks</td>
</tr>
<tr>
<td>New reqs</td>
<td>247</td>
<td>134</td>
</tr>
<tr>
<td>Req$ per week</td>
<td>6</td>
<td>45</td>
</tr>
</tbody>
</table>

- Walkthroughs led to more testable requirements that often replaced less adequate requirements from other acquisition techniques - Facilitators

- Planning further qualitative analysis
  - Blind expert judgement of randomly-selected requirements from scenario walkthroughs and other techniques
Where Requirements Came From

Was domain-specialisation effective?

- Walking through domain-specific scenarios will generate more requirements than domain-independent scenarios

<table>
<thead>
<tr>
<th>Event type</th>
<th>Normal</th>
<th>Generic alternative</th>
<th>Domain specific alternative</th>
<th>All types</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of equations</td>
<td>51</td>
<td>79</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

- Paired t-test on scenarios $p<0.002$, refute hypothesis H7

<table>
<thead>
<tr>
<th>Alternative course</th>
<th>Total in scenarios</th>
<th>%age of total scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>240</td>
<td>788</td>
</tr>
<tr>
<td>Domain specific</td>
<td>646</td>
<td>212</td>
</tr>
</tbody>
</table>

- Stated possible reasons include poor expression of some domain-specific alternative courses
Where Requirements Came From

Exploring normal course events
- All requirements linked to a normal course event

<table>
<thead>
<tr>
<th>Action type</th>
<th>Normal</th>
<th>Generic Alternative</th>
<th>Domain Alternative</th>
<th>All</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commun.</td>
<td>16</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cognitive</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Complex</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>System</td>
<td>35</td>
<td>40</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>51</td>
<td>79</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- Cross-tab analysis indicates variable independence
- Strong focus on human-machine interaction and functional requirements in scenarios
- Facilitation guidelines strongly influence resulting requirement types and triggering events
Taking the Process Forward

Overall results

- CREWS-SAVRE walkthroughs were central element of scenario-driven requirements engineering process
- Sufficient for follow-on contracts to support 3 further Eurocontrol projects using an improved process

Improving the process

- More cost-effective scenario generation
- Scenarios can be edited during walkthroughs
- More widespread adoption of use cases and scenarios, to make design decisions more explicit, and to provide common ground during workshops
- More effective scenario.Requirement linkage
- A more usable scenario walkthrough tool
Part 11: The VOLERE Requirement
Shell
What is a Requirement?

Definitions of a requirement

- Something that a product must do or a quality that the product must have (Robertson & Robertson 1999)
  - Interesting focus on product rather than software system
- Expression of the required phenomena that are shared between a machine (product) and the domain or environment (Jackson 1995)
  - Requirement refers to both the domain and the machine
- Requirements invariably contain a mixture of problem information, statements of system behaviour and properties, and design and implementation constraints (Sommerville and Sawyer 1997)
  - Requirements can be nasty little things to express
The Structure of a Requirement

Use the VOLERE template
- Available from http://www.atlsysguild.com
- See also standards such as PS-005 (Mazza et al. 1994)

<table>
<thead>
<tr>
<th>Requirement:</th>
<th>Requirement Type:</th>
<th>Event/use case:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source:</td>
<td>Fit criterion:</td>
<td></td>
</tr>
<tr>
<td>Customer satisfaction:</td>
<td>Customer dissatisfaction:</td>
<td>Conflicts:</td>
</tr>
<tr>
<td>Dependencies:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supporting materials:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Supermarket Checkout System Example

Important attributes
  – Type: Functional requirement
  – Description: The system shall print a receipt for each customer
  – Rationale: Customers need receipts as evidence...
  – Source: Store management team
  – Fit Criterion
    – For a representative sample of 100 customers and 1000 customer purchases with cash, debit cards, credit cards, cheques, club cards and special offer vouchers, the system prints the correct receipt with the correct information for 100% of all purchases. An independent human checker will test compliance by random sampling of 100 customers purchasing a representative range of items during one 8-hour test period, checking purchased items against items on the receipt.
  – Satisfaction: 3    Dissatisfaction: 5
  – Dependencies: Electronic cashier system
  – Supporting materials: Supermarket document XYZ.
Another Supermarket Checkout Example

- **Type:** Performance requirement
- **Description:** The system shall print all receipts within a time acceptable to the customer
- **Rationale:** Customers get impatient if kept waiting
- **Source:** Customer representatives
- **Fit Criterion**
  - For a representative sample of 100 customers and 1000 customer purchases the system prints all receipts of less than 25 items in less than 3 seconds, receipts of 26-40 items in less than 5 seconds, and 99% of a random sample of all receipts in less than 10 seconds from pressing the 'transaction complete' button. An independent human checker will test compliance by random sampling of 100 customers purchasing a representative range of items during one 8-hour test period. Timings will be tested using an electronic stopwatch based on observation.

  - **Satisfaction:** 4  
  - **Dissatisfaction:** 4

- **Dependencies:** Previous Functional Requirement
- **Supporting materials:** Supermarket document XYZ.
Part 12: The Requisite Pro Software Tool, and The VOLERE Quality Gateway
Requisite Pro - The Basics

Requisite Pro has a simple architecture
- Tight integration of MS Word and Access/SQL Server

- Requirements gathering
- Requirements description
- Requirements validation
- Requirements management
- Network facilities

- Requirements entering/downloading
- Attributes/structures/matrices
- Syntactic checking
- Versions/histories/change requests
- Client-server, web-enabled access

- Commercial data base
- Stores requirements/information
- Configuration management
Requirements Management Functions

Most requirements management tools will

- Trace links between requirements and other information
- Notify stakeholders of requirements changes
- Configure requirement versions
- Support requirement elaboration and refinement
- Produce documents and reports (ad infinitum!)
- Support simple queries about requirements
- Support data dictionaries/glossaries
- Limited requirements gathering
- Support requirements standards
Contribution Structures

Tracing requirements remains problematic
- Hard to locate and access
  - Human sources of requirements
  - Requirements-related information
  - Requirements-related work
- Contribution structures (Gotel & Finkelstein 1995) give a useful social structure
- Link agents to artefacts

Diagram:
- Principal Author
- Nominal Author
- True Author
- Scribe
- Devisor
- Ghost Author
- Representative Author
- Sponsor
- Documentor
Part 13: Writing Measurable Fit Criteria for Requirements
Why Measurable Requirements?

Strive for measurable requirements
- Making them potentially testable removes ambiguities
- Requirements can be testable at the contract stage or the implementation stage
- Improves understanding of the requirement by forcing more answers to questions about the requirement

So how to make requirements measurable
- Requirements types guide understanding
- Each requirement type determines the type(s) of units of measure available to write the measurable fit criterion
- Two-way process - requirement types suggest units of measure, and units of measure suggest requirements
- Demands a well-established requirements taxonomy
**Functional Requirements**

The most common type of requirement
- Something *(service, behaviour or function)* that a product must do (Robertson & Robertson 1999)

Examples are simple to find
- **Supermarket**: The system shall print a receipt
- **LAS-CAD**: The system shall detect the location of all ambulances
- **CORA-2**: The system shall present all candidate resolutions to the air traffic controller

How to test whether a functional requirement is met
- Measurable fit criterion in terms of *predicted outcomes*
- The receipt is *printed*, location of all ambulances are *known*, controller has *access to* all resolutions
Non-Functional Requirements

Also known as Quality requirements

– Express the desirable qualities of the product
– Predefined set of types: Performance, look-and-feel, device, usability, training, availability, maintainability, recoverability, portability, reliability, security, safety, contract and supplier-type requirements (based on Roman 1985, Robertson & Robertson 1999)
– Predefined types of units of measure and test, by type

Performance requirement

– Specifies time to do things, required throughput rates
– Measure using response times or times to undertake an action (speed), actions per time period (throughput), or numbers of units handled (load)
– Test using response timing, load tests, throughput tests
More Non-Functional Requirements

Look-and-feel requirement
- Specifies how end-users will perceive the product
- Measure using adherence to specified standards, use of colours, adoption of designs
- Test using observations of the product by independent assessors, standards compliance rules

Device requirement
- Specifies features, perhaps interactive, of the product
- Measure using adherence to specified standards, device features and attributes
- Test using observations of the product by independent assessors, standards compliance rules
More Non-Functional Requirements

Usability requirement
- Specifies how people will interact with the product
- Measure using task completion times, usage error-rates and usage rates and frequencies
- Test (with HCI techniques) with usability evaluation, protocol analysis and cognitive walkthrough techniques

Training requirement
- Specifies levels and nature of training to use the product
- Measure using training duration and outcomes
- Test using training course outcomes

Availability requirement
- Specifies nature and levels of access to the product
- Measure using times when available, %age downtimes
- Test using system-level trials and user experiences
More Non-Functional Requirements

Maintainability requirement
- Specifies the acceptable levels of upgrade of product
- Measure using time and resources to maintain
- Test using simple maintenance tasks

Recoverability requirement
- Specifies the repair of the product in case of failure
- Measure using time and likelihood to recover
- Test using simple maintenance and recovery tasks

Portability requirement
- Specifies platforms that product needs to operate on
- Measure using the names and versions of products and operating systems
- Test using usage trials on range of products
More Non-Functional Requirements

Reliability requirement
- Specifies levels of failure supported in the product
- Measure using mean-time between *(defined)* failures
- Test using product reliability trials, customer evidence

Security requirement
- Specifies levels of illegal access to the product
- Measure using specified access functions, and mean-time between breaches
- Test using security experts

Safety requirement
- Specifies how safe is the product
- Measure using number/risk of injuries in total/over time
- Test using health and safety compliance techniques
An Example Requirement

Look-and-feel requirement

- **Description**: The passengers’ open-door buttons shall be accessible to all passengers
- **Criterion**: The button is recognisable and accessible to people whose height is equal or above the average height of 5-year old UK child (x cms), adults in wheelchairs, people who are partially-sighted or registered-blind or registered disabled

Could have gone further...

- Could have gone to define how to test the requirements - possibly through different passenger-type requirements (decomposition), each with own fit criterion
Another Example Requirement

Maintenance requirement

- **Description:** The tube door system shall be maintainable and allow an engineer to readily access the electrical and mechanical mechanisms.

- **Criterion:** Using experience of systems in operation, and design alternatives, a set of common problems should be accumulated by the LU engineers. This set of problems should be further decomposed/ordered based on the average turnaround time to repair. An engineer should be able to reach and fix all known/common problems within the average turnaround times as specified. It is inevitable that a new system will bring new problems, and repair times for these should be based on past engineer experience. An engineer should be able to locate a fault within 15 minutes of inspection, and fix it within a further 2 hours.
And Another Example….

Reliability requirement

- **Description**: The system shall function as required in a dusty or smoky atmosphere
- **Fit Criterion**: The system shall function correctly within an atmosphere below the maximum curve in the graph below, showing dust tolerance for the system
Exercise: Practice Writing Measurable Fit Criteria for an Automated Teller Machine
Requirements: Automated Teller Machine

Purpose
- To practice writing measurable requirements according to the VOLERE requirement shell

Group task
- Discover and write some requirements for an automated teller machine that dispenses cash to customers using currently-available banking technologies
- Try to write requirements with a range of types, including functional, performance, look-and-feel, device, usability, training, availability, maintainability, recoverability, portability, reliability, security, safety, contract and supplier-type requirements

Assess
- What further information would you need to complete this task satisfactorily
Part 14: Linking Requirements to Use Cases
Use Cases, Requirements and Environment

Use cases
- Relate a system’s requirements to its environment

One downside
- It forces us to system boundaries
- Links through to concurrent requirements and creative design work
Linking Use Cases to Requirements

Express these links as a data meta-model

Each requirement specified at one of three levels

- **System**-level requirement
- **Use case**-level requirement
- **Action**-level requirement
Supermarket Checkout Use Case Example

Normal course

1. The operator swipes the customer's club card.
2. The system reads customer information from club card.
3. **REPEAT** while more products to be purchased.
   3.1 The operator swipes the product using the barcode reader.
   3.2 The system displays the item name and price.
   3.3 The system records purchase details from the product barcode.
4. The operator presses the 'transaction total' button.
5. The system displays the total amount due for the purchase.
6. The customer pays the operator.
7. The operator returns change to the customer.
8. The operator then presses the 'transaction paid' button.
9. The system records the entire transaction.
10. The system updates the customer's clubcard details with the purchase information.
11. The system prints a receipt with all purchase details on the receipt.
Supermarket Checkout Example Ctd....

Alternative courses

1. The customer might not have a club card.
2. The customer might not have a club card.
3. The customer can ask for the price of an article.
3.1. The item might not have a bar code, e.g. fruit and vegetables. The operator places the item on the scale. The operator enters a 4-digit number which is a unique identifier for the fruit or vegetable. The operator presses the button to calculate the price.
3.1 If the bar code reader cannot read the product code, the actor must enter this code using the keyboard;
3.1 If one product is bought several times, the operator can swipe the bar code ones, and multiply the price by the number of products.
3.1 The operator can cancel the last bar code swiping operation.
3.1 The operator can deduce a special reduction from the price of an item.
3.3. The item might not have a bar code. The system records purchase details from weight information from the scales and the unique identifier entered by the operator.
4. The system subtracts from the total amount the club reductions.
Decomposing Requirements

Requirement **DECOMPOSES INTO** Use Case

- The requirement is satisfied if the systems and other agents undertake all of the behaviour in the use case

Supermarket checkout example

Requirement:
The system shall enable operators to manage the purchase of items from the supermarket

Decomposed into

Use case: Purchase items
Constraints Imposed by Requirements

Requirement CONSTRAINS Use Case

- The requirement is satisfied if the systems and other agents undertake the behaviour expressed in the use case within the constraints expressed in the requirement

Supermarket checkout example

Requirement:
The system shall enable the completion of 95% of customer purchases within 90 seconds of the start of the purchase of the first item

Use case: Purchase items
Requirements Enable Behaviour

Requirement **ENABLES** Action

- The requirement is satisfied if the requirement enables the system and/or actor to undertake the behaviour described in the use case

Supermarket checkout example

2. The system reads customer information from club card.
   FR1: The system shall read a standard club card barcode.

3. **REPEAT** while more products to be purchased.
   3.1 The operator swipes the product using the barcode reader.
      TR1: The operator shall be trained to read the barcode reader.
      DR1: The system shall have graphical instructions to use the reader.
   3.2 The system displays the item name and price.
      FR2: The system shall have a standard product barcode.
      FR3: The system shall retrieve product details from the product data base.
      FR4: The system shall display product name and price on the display.
Requirements Constrain Behaviour

Requirement \textbf{CONSTRAINTS} Action

- The requirement is satisfied if the system and/or actor undertakes the behaviour described in the use case within the constraints expressed in the requirement

Supermarket checkout example

2. The system reads customer information from club card.
   - \textbf{PR1:} The system shall read all barcodes within 0.25 seconds.
   - \textbf{RR1:} The system shall read the code of 9999/10000 of swiped barcodes.

3. \textbf{REPEAT} while more products to be purchased.
   3.1 The operator swipes the product using the barcode reader.
   3.2 The system displays the item name and price.
   - \textbf{RR2:} The system shall read the code of 9999/10000 of swiped barcodes.
   - \textbf{PR2:} The system shall read all barcodes within 0.01 seconds.
CORA-2: Specifications and Walkthroughs

CORA-2 operational requirements document
- All CORA-2 requirements structured around 22 essential use cases for the CORA-2 system

**System-level requirements**

The CORA-2 system shall...

**Use case: Generate Resolutions**

The CORA-2 system shall...
The CORA-2 system shall...

**Generate Resolutions Action 1**

The CORA-2 system shall...
The CORA-2 system shall...
The CORA-2 system shall...

**Generate Resolutions Action 2**

The CORA-2 system shall...
The CORA-2 system shall...
The CORA-2 system shall...
The CORA-2 system shall.....
Exercise: Discovering Requirements with Use Cases for the Security Access System
Exercise: Security Access System

Use case: Security access system to building
1. The user swipes card through magnetic swipe.
2. The system validates the card and its access rights.
3. The system unlocks the door.
4. The user opens the door.
5. The user passes through and closes the door.
6. The system locks the door.

Task: Specify requirements for system
- Discover requirements from this normal course use case, and integrate them into use case structure
- Discover requirements from possible alternative courses, and integrate them into use case structure
- Make all requirements testable
Part 15: Sources of Expertise and Knowledge for Measurable Fit Criteria
CORA-2 Difficulties

Difficulties writing measurable fit criteria

- Lack of domain experts available to provide input to the requirements specification process
- Lack of expert commitment to provide numbers for measurable fit criteria
- No repository of expertise for making judgements that can lead to commitment
- Specification process stalled at this point

So what can be done?

- Locate and involve internal and external experts throughout the RESCUE process
- Start to construct libraries of externalised knowledge with which to write measurable fit criteria
Satisfaction Arguments

Satisfaction arguments can provide help here
  – From Michael Jackson research on requirements
  – Applied in the REVEAL process from Praxis Critical Systems

What are satisfaction arguments
  – Structured rationale for a requirement and how it will be satisfied by the future system