STANDARDS: ESA PSS-05

Introduction to the UR Phase

Feliz Gouveia

Software Engineering

The UR Phase

- The UR phase can be called the ‘problem definition phase' of the life cycle.
- The phase refines an idea about a task to be performed using computing equipment, into a definition of what is expected from the computer system.

Capture of UR

- user requirements should be clarified through criticism and experience of existing software and prototypes;
- wide agreement should be established through interviews and surveys;
- knowledge and experience of the potential development organisations should be used to help decide on implementation feasibility, and,
  perhaps to build prototypes.

UR should be realistic

- Realistic user requirements are:
  - clear;
  - verifiable;
  - complete;
  - accurate;
  - feasible.

Requirements

- Clarity and verifiability help ensure that delivered systems will meet user requirements.
- Completeness and accuracy imply that the URD states the user's real needs.
- A URD is inaccurate if it requests something that users do not need, for example a superfluous capability or an unnecessary design constraint

Realistic UR

- Realistic user requirements must be feasible. If the resources and timescales available for its implementation are insufficient, it may be
unrealistic to put them in a URD.

Operational environment

- A clear description of the real world that the software will operate in should be built up, as the user requirements are captured.
- This description of the operational environment must clearly establish the problem context

Users

- The roles and responsibilities of the users and operators of software should be established by defining the:
  - characteristics of each group (e.g. experience, qualifications);
  - operations they perform (e.g. the user of the data may not operate the software).

Capability requirements

- Capability requirements describe the process to be supported by software.
- Simply stated, they describe ‘what' the users want to do.
- A capability requirement should define an operation, or sequence of related operations, that the software will be able to perform.

Capability requirements (1)

- Quantitative statements that specify performance and accuracy attributes should form part of the specification of capability.
- This means that a capability requirement should be qualified with values of:
  - capacity;
  - speed;
  - accuracy.
- The performance attribute is the combination of the capacity and speed attributes.

Capacity

- The capacity attribute states ‘how much' of a capability is needed at any moment in time. Each capability requirement should be attached with a quantitative measure of the capacity required.
- For example the:
  - number of users to be supported;
  - number of terminals to be supported;
  - number of satellites that can be controlled simultaneously;
  - amount of data to be stored.

Speed

- The speed attribute states how fast the complete operation, or sequence of operations, is to be performed.
- Each capability requirement should be attached with a quantitative measure of the speed required. There are various ways to do this, for example the:
  - number of operations done per unit time interval;
  - time taken to perform an operation.
- For example: ‘95% of the transactions shall be processed in less than 1 second', is acceptable whilst, ‘95% of the transactions will be done as soon as possible' is not.

Constraint requirements

- Constraint requirements place restrictions on how the user requirements are to be met.
- The user may place constraints on the software related to interfaces, quality, resources and timescales.
- Users may constrain how communication is done with other systems, what hardware is to be used, what software it has to be compatible with, and how it must interact with human operators.
- These are all interface constraints. An interface is a shared boundary between two systems; it may be defined in terms of what is exchanged across the boundary

Communications interfaces

- A communications interface requirement may specify the networks and network protocols to be used. Performance attributes of the interface may be specified (e.g. data rate).

Software interfaces

- A software interface requirement specifies whether the software is to be compatible with other software (e.g other applications, compilers, operating systems, programming languages and database management systems).

Human-Computer Interaction

- A Human-Computer Interaction (HCI) requirement may specify any aspect of the user interface. This may include a statement about style (e.g. command language, menu system, icons), format (e.g. report content and layout), messages (e.g. brief, exhaustive) and responsiveness (e.g. time taken to respond to command). The hardware at the user interface (e.g. colour display and mouse) may be included either as an HCI requirement or as a hardware interface requirement.

Availability

- Availability measures the ability of a system to be used during its intended periods of its operation. Availability requirements may specify:
  - mean and minimum capacity available (e.g. all terminals);
  - start and end times of availability (e.g. from 0900 to 1730 daily);
  - time period for averaging availability (e.g. 1 year).
- Examples of availability requirements are: ‘the user shall be provided with 98% average availability over 1 year during working hours and never less than 50% of working hours in any one week';

Portability

- Software portability is measured by the ease that it can be moved from one environment to another. Portable software tends to be long lived, but more code may have to be written and performance requirements may be more difficult to meet.
- An example of a portability requirement is: ‘the software shall be portable between environments X and Y'.

Security

- A system may need to be secured against threats to its confidentiality, integrity and availability.
- For example, a user may request that unauthorised users be unable to use the system, or that no single event such as a fire should cause the loss of more than 1 week's information.
- The user should describe threats that the system needs to be protected against, e.g. virus intrusions, hackers, fires, computer breakdowns.
- The security of a system can be described in terms of the ownership of, and rights of access to, the capabilities of the system.

Safety

- The consequences of software failure should be made clear to developers.
- Safety requirements define the needs of users to be protected against potential problems such as hardware or software faults.
- They may define scenarios that the system should handle safely (e.g. ‘the system should ensure that no data is lost when a power failure occurs')

Standards

- Standards requirements normally reference the applicable documents that define the standard.

Resources

- The resources available for producing and operating the software are a constraint on the design.
- Resource requirements may include specifications of the computer resources available (e.g. main memory). They may define the minimum hardware that the system must run on.

Acceptance Test Planning

- Validation confirms whether the user requirements are satisfied when the software is delivered.
- This is done by performing acceptance tests in the Transfer Phase.

UR Review

- The outputs of the UR phase must be formally reviewed during the User Requirements Review. This should be a technical review.
- Normally, only the URD and the Acceptance Test Plan undergo the full technical review procedure involving users, developers, management and quality assurance staff.

Methods for UR definition

- Interviews and surveys
- Observation of existing systems
- Prototyping
- Feasibility studies
- …

Output of this phase: URD

- A URD is ‘clear' if each requirement is unambiguous and understandable to project participants.
- A requirement is unambiguous if it has only one interpretation. To be understandable, the language used in a URD should be shared by all project participants and should be as simple as possible

The URD

- Each requirement should be stated in a single sentence. Justifications and explanations of a requirement should be clearly separated from the requirement itself.
- Clarity is enhanced by grouping related requirements together. The capability requirements in a group should be structured to reflect any temporal or causal relationships between them.

The URD (1)

- A URD is consistent if no requirements conflict. Using different terms for what is really the same thing, or specifying two incompatible qualities, are examples of lack of consistency.

The URD (2)

- A URD is modifiable if any necessary requirements changes can be documented easily, completely, and consistently.
- A URD contains redundancy if there are duplicating or overlapping requirements.
- Redundancy itself is not an error, and redundancy can help to make a URD more readable, but a problem arises when the URD is updated.

The URD (3)

- Changes to the URD are the user's responsibility. The URD should be put under change control by the initiator at soon as it is first issued.
- This requires that a change history be kept.
- New user requirements may be added and existing user requirements may be modified or deleted. If anyone wants to change the user requirements after the UR phase, the users should update the URD and resubmit it to the UR/R board for approval.

The URD (4)

- The definition of the user requirements must be the responsibility of the user.
- This means that the URD must be written by the users, or someone appointed by them.
- The expertise of software engineers, hardware engineers and operations personnel should be used to help define and review the user requirements.

Acceptance Test Plans

- The initiator(s) of the user requirements should lay down the acceptance test principles.
- The developer must construct an acceptance test plan in the UR phase and document it.
- This plan should define the scope, approach, resources and schedule of acceptance testing activities.
- Specific tests for each user requirement are not formulated until the DD phase.

Acceptance Test Plans (1)

- The Acceptance Test Plan should deal with the general issues, for example:
  - where will the acceptance tests be done?
  - who will attend?
  - who will carry them out?
  - are tests needed for all user requirements?
  - what kinds of tests are sufficient for provisional acceptance?
  - what kinds of tests are sufficient for final acceptance?
  - must any special test software be used?