A Unified Approach to System Functional Testing

Paul Gerrard
Gerrard Consulting Limited
PO Box 347
Maidenhead, SL6 2GU, UK
Tel: +44 (0)1628 639173
paul at gerrardconsulting dot com
www.gerrardconsulting.com

Abstract

Requirements are the foundations of every project yet we continue to build systems with requirements that have not been tested. We take care to test at every stage during design and development and yet the whole project may be based on untested foundations.

Functional system tests should be based around coverage of the functionality described in the requirements, but it is common for the design document to be used as the baseline for testing because the requirements can't be related to the end product. In the worst case, system tests can become large scale repetitions of unit tests. It is not surprising that many system tests fail to reveal requirements errors.

We ask users to perform acceptance tests against their original requirements. But who can blame enthusiastic users when they become overwhelmed by the task? The system bears so little resemblance to what they asked for that the acceptance test often becomes a superficial hands-on familiarisation exercise.

This paper proposes that a unified view of requirements can improve the requirements gathering process, give users a clearer view of their expectations and provide a framework for more effective system and user acceptance tests.

Prerequisite Key Words: none

Topic Descriptors: Requirements Testing, System Testing, User Acceptance Testing
1. INTRODUCTION

1.1. Poor Requirements

It is generally accepted that communications between users and developers are a problem. Different perspectives tend to cause both parties to misinterpret and misjudge the importance of the issues at stake when laying the foundation of systems development projects.

Users and system developers typically have different objectives. Users want the right system, developers want to build the system right. The lack of a common business language frustrates the production of a high quality statement of requirements but this is one of many factors contributing to the difficulties. Most of these difficulties can be traced to the significant cultural differences between the two sides.

The disastrous impact of getting only a few requirements wrong at the outset may not be revealed until acceptance or, even worse, in production. In some projects, requirements may never be frozen at all and projects drift steadily towards calamity.

Requirements documents rapidly lose relevance if requirements gathered on the fly are catered for in the design, but never recorded. Requirements documents are often badly put together - it is common for omissions, inconsistencies and errors to be ignored. After all, "once the design is finished, we don't need them anyway".

When the time comes for system testing, badly written requirements can be impossible to organise into a useful baseline for use in the test, so the obvious fallback is to use the design document instead. It should come as no surprise when system tests fail to reveal errors in requirements.

We ask users to undertake acceptance by testing the delivered system against their original requirements. From the users' simple view of what was required originally, the requirements grew into a voluminous tome, which they thought they had seen the last of months or years earlier. Systems usually bear little resemblance to the requirements as signed-off.

To get the users going, they may then be given an inventory of the screens and reports in the system which they then use as a check list. In the worst case, acceptance becomes an extension to the training they may have received the week before.

If this is a common scenario, is it any surprise systems fail to meet the users' expectations?
1.2. Economics of Testing Requirements

Tales of woe described above are unfortunately all too common. Surely we could use testing more effectively in the requirements gathering stage to reduce the problem?

Advocates of good test practice have been saying for some years that testing accounts for around half the cost of a typical project. It is generally accepted that requirements errors, being the most expensive might comprise 80% of the cost of all errors.

If we were to allocate test resources in proportion to the potential cost of errors found, we would naturally allocate 40% of our project budget to testing the requirements. The reality is somewhat different. Usually, the only ‘test’ applied to requirements is an informal review by users.

The economics seem to argue for starting testing as early as possible - but how can we test requirements?

1.3. Requirements Strategy

Requirements represent the interface between user and developer. Although this interface is most critical in the early stages of a project, failures only have their disastrous impact towards the end. We adopt a requirements strategy as follows:

- adopt a testable notation for expressing testable user expectations
- test requirements
- validate user expectations
- base system tests on requirements
- base acceptance tests on user expectations

Only by documenting and verifying the user requirements can we build systems to meet those requirements. Only if we document and validate the users' expectations can we hope to demonstrate that those expectations have been met.

1.4. V-Model

Can we fit such a scheme into the V-Model? The V-Model viewpoint suggests we test at each stage verifying against baseline documentation. The problem with requirements documents is that there is no baseline. One could say the baseline is in the users' heads - it is their expectation.

The V-model focuses on dynamic tests only. The problem is that it is a technically oriented developers' view of the process. We need to adopt a more user or requirements-oriented viewpoint.
1.5. Pawnbrokers Model of Requirements

Here is a simple model of requirements - we've split them into required, designed and delivered requirements. It is useful because it doesn't place the emphasis on the development process.

![Figure 1 Pawnbroker Model of Requirements](image)

Our aim, obviously, is to make the three regions coincident and maximise the area at the centre. It is interesting to consider the situations of requirements falling outside the central area and to imagine how such situations arise:

- required, designed, not delivered (cuts through deadline pressure?)
- not required, designed, delivered (an analyst's bright idea?)
- required, not designed, delivered (the user nobbled a programmer?)

How does testing fit with our simple model? It is clear that current testing practice focuses attention on the overlap of design and delivery.

![Figure 2 Current Testing Practice and the Pawnbroker Model](image)
The reasons for this imbalance are mainly attributable to our technical prejudices:

- requirements gathering is a hit and miss affair which can't be systematised
- users often don't know what they want until they are asked to test
- informal requirements can't be tested against
- formal methods are the only solution anyway

This paper proposes that by taking a unified view of requirements we can dispel these prejudices.

2. A UNIFIED APPROACH

2.1. Objectives

In adopting a unified approach, we set four objectives:

- an up-front agreed definition of requirements that users will commit to and developers will adhere to and that both parties understand
- to cast users' expectations of the proposed system in terms of the requirements (not in spite of them)
- to define requirements in terms that make it possible to validate them
- to demonstrate that the users' expectations have been met.

We aim to do this by using a single framework for requirements, system and user tests.

2.2. Behavioural Model of Requirements

A behaviour model is a model of the users expectations of a system. It has been proposed that user requirements could be stated entirely in terms of behaviours. For current purposes, we will assume a standard requirements document already exists.

Behaviours are derived from an analysis of the requirements document. Each behaviour has four components:

- a function description
- an event that triggers the function
- the condition(s)
- the response of the system predicted by the requirements.

Superficially, a behaviour is similar to a simple test case, but its purpose is to define the user expectation for a feature of the system under defined conditions.
2.3. **Summary of Process**

The process is integrated with the normal development phases and is illustrated in the table below:

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>Requirements Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>During Requirements Gathering</td>
<td>Analysis of the requirements document to derive system behaviours</td>
</tr>
<tr>
<td></td>
<td>The requirements are tested by a structured review of the behaviours</td>
</tr>
<tr>
<td></td>
<td>Behaviours are assigned a test coverage target</td>
</tr>
<tr>
<td></td>
<td>Business scenarios are derived which 'cover' the behaviours</td>
</tr>
<tr>
<td></td>
<td>We animate the requirements allowing users to verify the behaviours</td>
</tr>
<tr>
<td>During System Test Planning</td>
<td>System and user acceptance tests use the behaviours as test cases</td>
</tr>
</tbody>
</table>

*Table 1 Requirements Activities*

2.4. **Automation**

Modelling requirements as behaviours lends itself to automation. Simple record management is adequate for most purposes and it can be quite effective in the animation of requirements.

PC database tools allow us to capture the text of requirements directly into a database as sentences. We can structure the document by sub-system, module, feature. We can tag, cross-reference, prioritise requirements. We can define tables, import or link graphics.

Most databases can produce requirements documents directly from the database on laser printers. We can also be selective in printing - for example, we can query and print selections of requirements for each user to review. To control pagination, indexing and the like, for report production and distribution, the data can be exported into a word processor at any time.

Useful animations do not require very sophisticated software. A screen allowing the user to scroll through business scenarios displaying the scenario stage instructions and the system behaviour is very effective. If the user can approve or reject the behaviour, or make other comments, even better.
3. PROCESS

3.1. Deriving Behaviours

The process of deriving behaviours from requirements documents is fairly mechanical. Section by section the document is scanned for functions, conditions and responses and lists are made of each. Every item listed is referenced to a requirements sentence.

For every function each condition identified is considered. If the condition is relevant to the function:

- if possible, a response is selected from the list
- responses not explicitly identified are assumed or left blank then tagged
- responses which seem inconsistent or erroneous are tagged

The analyst should consider combinations of conditions which are possible, but may not have responses predicted. These are added to the table of behaviours.

Conditions identified in requirements most often represent equivalence partitions - the analyst can add others if they seem appropriate and so provide more explicit definitions of behaviours for the user to verify.

The process of deriving behaviours from a requirements document is an excellent inspection method. The analyst will tabulate the behaviours and as a result, missing and inconsistent responses usually stand out.

3.2. Testing Requirements

The system behaviours are presented to users in the form of a table. By tagging system features with user roles, it is possible to be selective, and give users only those behaviours that are relevant. Users can informally review each set of behaviours individually, but a more formal review is preferable.

In a structured review, the analyst asks the user to verify each behaviour in turn. The analyst can emphasise the behaviours that have been tagged and draw these to the users' attention. As a rule most behaviours can be checked in a few seconds when presented in this way. Tagged items can often be resolved on the spot.

This process encourages the user to think in terms of their interaction with the system and to compare the responses with their assumptions. It is likely that features and conditions that the user assumed existed, are spotted and added now.

The results of these reviews are consolidated, the behaviours updated and the requirements document republished together with the behaviours table.
3.3. **Coverage Targets**

At the review stage, or as a later activity in consultation with user management, test coverage targets can be set for each behaviour. Criteria for this could be:

- features that are critical to the successful implementation of the system
- features that are anticipated to be complex to implement
- features that past experience shows are error prone

Behaviours might be assigned a coverage target according to the scheme below:

- covered by at least one test case
- covered by several test cases complemented by cases derived from boundary value analysis
- treated as a special case and test cases will be derived using formal techniques.

3.4. **Business Scenarios**

Users are asked to invent business storylines which reflect common business situations. These simulate the context in which the system will be used. Common types of scenarios are:

- life histories of key business entities e.g. customers, products, people etc.
- projects e.g. build an oil platform, launch a product, deliver a large order
- business cycles: daily, weekly monthly, annual etc.

These scenarios are split into scenario stages. Each stage represents a business event or activity which is then mapped to a behaviour. As the scenarios are created, it is possible to assess the coverage of each behaviour by counting the scenario stages that reference it. New scenarios or variations of existing scenarios are created to achieve the coverage target.

3.5. **Animation of Requirements**

Business scenarios and the behaviours they reference provide an integrated view of a business process and the system behaviour. Taken together, they model the users expectations of the system in its business context. The scenarios represent typical (and untypical) business situations in which the system will be used.

The scenarios provide information relating to the business context which the user can relate to. The behaviours referenced by the scenario stages describe the system features to be used, the conditions and the expected responses of the system. A paper printout of the scenarios with the behaviour information provides a script for a structured walkthrough of how users will interact with the system in production. This is a very effective way of verifying the requirements in a business context.

If the behaviours and scenarios are held in a database, it can be used to animate the requirements. Assembling teams of business people for such intensive reviews can be difficult, so it may be more practical for individuals to use the animation. The same
animation can be repeated for different users, or departments and can be used to test both the requirements and any revised business process.

Using the database, the user is invited to step through each stage in a scenario in turn and approve or reject the proposed behaviour. The user can make comments at any stage. It is possible and useful to step backwards through stages as well as forwards. The animation gives the user a clearer picture of how the system will behave in production and confirms their expectations of the system.

We can also view the animation as a rehearsal of the user test as the scenarios will be used to generate the test scripts later on.

3.6. Test Planning

The database of scenarios and behaviours can be used to generate draft test scripts. The scenario provides test script sequence, step definition, and business instruction. The behaviour defines the function to perform, the condition or inputs required and the expected response.

The requirements animation took the user through a business scenario which referenced at each stage a system feature, the current conditions and the system's response to the situation. The animation of the business scenario can be seen as a rehearsal of the system or user acceptance test also. This can help the planning and organisation of the real test.

During requirements gathering, all the conditions significant to the user functions have been captured. Test coverage targets were set at the same time, while the issue was foremost in the users mind. Business scenarios were selected to achieve the required coverage. Using the business scenarios in this way guarantees that our functional test coverage targets are achieved. All that remains is for detailed instructions for the execution of system transactions to be derived and added to the draft test scripts.

4. CONCLUSION

A unified view of requirements helps us to obtain more reliable requirements statements. Modelling user requirements in terms of system behaviours achieves this unification. It allows us to test requirements and helps us to achieve completeness, consistency, accuracy and relevance in requirements documents.

Animation of requirements using even a simple tool can help to ensure users have clear expectations of a proposed system in harmony with the developers commitment. System behaviours can be reused as a framework for both system and user acceptance tests. They allow coverage targets to be set and met as well as facilitating the planning, preparation and organisation of the tests themselves.

A unified view of requirements complements the requirements gathering process, can enhance the users' involvement, and improve the chances of meeting their expectations.