Synthesising Test-based justification of Problem Oriented Software Engineering

A Thorpe

16 September, 2008

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Synthesising Test-based justification of Problem Oriented Software Engineering

A dissertation submitted in partial fulfilment of the requirements for the Open University’s Master of Science Degree in Software Development

Adam Thorpe

8 March 2009

Word Count: 14,783
Preface

I would like to extend my gratitude to all those who have given their support and advice during the course of this dissertation. Particularly my project supervisor Brian Kavanagh, and my technical advisor Jon Hall. Lastly, to my wife Antonella, and two daughters Paige and Isabelle, for their patience and understanding.
Table of Contents

Preface ........................................................................................................................................1

List of Figures .........................................................................................................................viii

List of Tables ..............................................................................................................................ix

Chapter 1 Introduction ...........................................................................................................1

1.1 Background to the research ...........................................................................................1

1.1.1 Requirements Engineering ......................................................................................1

1.1.2 Problem Oriented Software Engineering ...............................................................2

1.1.3 Testing .......................................................................................................................3

1.1.4 Software Development Life-Cycle ..........................................................................3

1.1.5 Quality Assurance ....................................................................................................4

1.2 Aims and objectives of the research project ..................................................................4

1.3 Summary ..........................................................................................................................6

Chapter 2 The Problem in Context .....................................................................................8

2.1 Introduction ......................................................................................................................8

2.2 Problem Oriented Software Engineering .....................................................................8

2.2.1 POSE Features ........................................................................................................9

2.2.2 POSE Problem .........................................................................................................10

2.2.3 POSE Nouns ............................................................................................................11
2.2.4 POSE Transforms.......................................................................................... 11

2.2.5 POSE Justification........................................................................................ 11

2.2.6 POSE Patterns ............................................................................................ 13

2.2.7 POSE Specification Examples ..................................................................... 17

2.2.8 Justification General Form ......................................................................... 18

2.2.9 Related Work ............................................................................................... 19

2.3 Formal Methods ............................................................................................... 19

2.4 User Centred Design ....................................................................................... 20

2.5 Software Development Life-Cycle .................................................................. 21

2.6 Requirements Engineering .............................................................................. 22

2.6.1 Fit Criteria ................................................................................................... 23

2.6.2 Quality Gateway ......................................................................................... 23

2.6.3 Functional Requirements ............................................................................ 23

2.6.4 Non-Functional Requirements .................................................................... 24

2.6.5 Constraints .................................................................................................. 24

2.7 Testing ............................................................................................................. 25

2.7.1 Testing categories ....................................................................................... 26

2.7.2 Testing Alternatives .................................................................................... 27

2.7.3 Testable Requirements ................................................................................. 28
2.7.4 Testing Documentation Standard ................................................................. 28

2.8 Project Management.......................................................................................... 29

2.9 Test Driven Development ............................................................................... 29

2.10 eXtreme Programming ................................................................................ 29

2.11 Software Quality ........................................................................................... 30

2.12 Research question ....................................................................................... 31

2.13 Summary ....................................................................................................... 31

Chapter 3 Research Approach ........................................................................... 32

3.1 Introduction .................................................................................................... 32

3.2 Research Techniques ..................................................................................... 32

3.3 POSE and Functional Requirements Testing ................................................ 33

3.4 POSE and Non-functional Requirements Testing ........................................... 33

3.5 Test Transforms for POSE ........................................................................... 34

3.6 Synthesis of Activities .................................................................................. 35

3.7 Alternative Approaches ............................................................................... 35

3.8 Summary ....................................................................................................... 36

Chapter 4 Designing POSE, Tests and Testing Transforms .............................. 38

4.1 Introduction ................................................................................................... 38

4.2 Writing the POSE specification ..................................................................... 38

4.2.1 The Problem ........................................................................................... 38

4.2.2 User Centred Design .............................................................................. 39
5.6 Testing ................................................................................................................. 57

5.7 POSE Transforms and Testing ............................................................................. 57

5.8 POSE Nouns and Testing .................................................................................... 59

5.9 POSE and Testing Strategies ............................................................................. 60

5.10 The Testing Role ............................................................................................... 60

5.11 Summary .......................................................................................................... 60

Chapter 6 Test-based Justification ........................................................................... 62

6.1 Introduction ......................................................................................................... 62

6.2 Levels of Test-based Justification ...................................................................... 63

6.2.1 Level 0 – Do nothing .................................................................................... 64

6.2.2 Level 1 – Fit Criteria ................................................................................... 64

6.2.3 Level 2 – Test Overview ............................................................................. 66

6.2.4 Level 3 – Test Design .................................................................................. 68

6.3 Designing Justifications .................................................................................... 69

6.3.1 Level 1 – Fit Criteria ................................................................................... 69

6.3.2 Level 2 – Test Overview ............................................................................. 70

6.3.3 Level 3 – Test Design .................................................................................. 70

6.4 Validating .......................................................................................................... 71

6.5 Architecture ....................................................................................................... 73

6.6 Review of Testing Transforms & Nouns as Recording Options ....................... 74
List of Figures

Figure 1 The POE process pattern .................................................................13

Figure 2 Sequential POE Process.................................................................14

Figure 3 Parallel POE Process .................................................................15

Figure 4 Fractal POE Process .................................................................16

Figure 5 Problem and Solution Exploration as Problems .........................17

Figure 6 Vee life-cycle model (taken from The Open University 2007) ........22

Figure 7 POSE Vee Process Life Cycle .....................................................53

Figure 8 Architectural POSE Vee Process Life Cycle ...............................54

Figure 9 Component POSE Vee Process Life Cycle .....................................55

Figure 10 POSE and the bigger picture ......................................................62

Figure 11 the Justification and Test Overlap .............................................67

Figure 12 The individual income tax return problem ................................93

Figure 13 Generalised form composition - a UML Class Diagram ...............97
List of Tables

Table 1 Summation of POSE Specification ................................................................. 41

Table 2 the Relationship between POSE transforms and Vee Testing Levels ............ 59

Table 3 Validation & Review Frequency ..................................................................... 72

Table 4 Involvement of Test and Solution Designers in TBJ ................................. 78

Table 5 User Characteristics ...................................................................................... 130

Table 6 Requirements derived from User Characteristics ........................................ 131

Table 7 Task Analysis for completing Form 1040 .................................................... 132

Table 8 Requirements from Task - complete Form 1040 ........................................ 134

Table 9 Requirements from initial workflow analysis .............................................. 135

Table 10 Usability Non-functional Requirements ...................................................... 136
Abstract

Problem Oriented Software Engineering (POSE) is a young framework supporting requirement and design specification. POSE allows a blend of formal and non-formal. Much of POSE research has been concerned with safety-critical systems, where a justification case is required by legislation.

Hall et al. (2007a) suggested that an alternative, and as yet undefined, method of justification based on testing may be cheaper than the existing approach. Also, to date there has been no research into the relationship between testing and POSE.

The project identifies an approach to test-based justification of POSE. I arrived at this through a synthesis of observations (based on writing a POSE specification and associated test designs), professional experience, and literary review. My approach has three incremental levels of justification detail, with each representing a decrease in risk, and an increase in cost, from the previous one. These levels are framed to describe the relationship between quality assurance, project management, development methodology and POSE.

A by-product of this work has been a clearer understanding of the relationship between POSE and Testing within the software development life-cycle.

This project is likely to be of interest to those using POSE for a development project, including quality assurance members, project managers, development managers, designers, testers, clients, and the POSE research community,
Chapter 1  Introduction

1.1 Background to the research

Problem Oriented Software Engineering (POSE) is a relatively young research area spanning requirements engineering and design, this is in contrast to testing which is a well understood discipline.

Currently in POSE, validation is performed through the use of justifications that stemmed from assurance-based development; I term these as assurance-based justifications (ABJ) from here. Hall (2007a) suggests that an alternative to ABJ would be to validate using test-based justification (TBJ). Validation by TBJ although less rigorous than ABJ, might be cheaper and therefore more popular among certain clients and problems. The dissertation is based on the hypothesis that TBJ can be achieved by extending POSE.

To date POSE research has not considered how the design and specification tests can be derived from a POSE specification, this dissertation aims to fill this knowledge gap.

1.1.1 Requirements Engineering

Requirements can be categorised as being functional, non-functional or constraint, (Robertsons 1999). Functional requirements specify the systems behaviour, while non-functional requirements specify the systems qualities, lastly constraints specify global requirements. Eliciting and specifying functional requirements is considered easier than non-functional requirements (Robertsons 1999). A feature of specifying non-functional requirements is that they tend to need further specification of context, and that they impact other downstream requirements. Currently the body of examples within POSE
contains purely functional requirements; there are no recorded non-functional requirements or constraints.

1.1.2 Problem Oriented Software Engineering

The intention of POSE was to create a framework for solving software problems to arrive at a solution specification (Rapanotti, et al. 2008 and Hall, et al. 2007a) that would allow:

- Information about the real world, to sit side by side with requirements and the solution to the requirement.
- To allow analysis of requirements.

The main element within POSE is the problem sequent, this is a three part mathematical structure $W, S \models R$. $W$ stands for the non-formal, the real world within which the problem must be solved, this is not always explicitly captured in other requirements engineering techniques. $S$ represents the solution to the problem, and $R$ is the requirement. We could consider the problem sequent as being the nouns that POSE manipulates.

The POSE specification is presented top down and allows drilling down into further depth in the real world, the solution or the requirement. This supports the notion of decomposition, and the graphical output of POSE is essentially an inverted tree-diagram with the root at the top.

Sequents are transformed according to a defined set of transforms, for example solution expansion. Each transform will yield a new sequent. The transforms can be viewed as the verbs of POSE.
The final piece of the POSE is the POE process pattern. This pattern describes an approach to designing systems; it contains problem and solution exploration as well as validation points.

To extend POSE to include test-based justification implies a need to make changes or additions to the POSE nouns, verbs and process.

1.1.3 Testing

Testing is an expensive activity (Beizer 1990) that can consume 50% of the resources of a software development project. The aim of testing strategy (Myers 2004) is to select the set of test cases that will have the greatest chance of revealing errors in the software. Myers demonstrates that exhaustive testing is not possible, therefore a sub-set of all possible test cases must be selected.

A simplification of testing is to view it a three part activity:

1. Determine test strategy.
2. Design test cases.
3. Execute test cases.

Functional testing will comprise of black box testing techniques, covered extensively by Beizer (1990).

There has been no research into testing in conjunction with POSE.

1.1.4 Software Development Life-Cycle

The Open University (2006a) shows the Vee Life-Cycle for software development; for each activity within the software development process, there is an associated testing level. These levels are unit, component, integration, system, and acceptance testing. Each level has a different objective, therefore writing the user-acceptance tests that this
dissertation will demand, will need to keep this objective in mind. POSE can be used to write specifications down to, but not including coding. Therefore there is a relationship between POSE design and all levels of testing other than unit testing.

1.1.5 Quality Assurance

Justification of specifications is a practice that is dictated by quality assurance. In some cases such as safety critical systems, it is legislation that is enforcing the use of safety cases, a form of justification. This form of legislation is an external factor, but it is still a form of quality assurance.

Software quality comes from two places 1) the requirements specification states the qualities that the product must provide 2) building the system using best practice approaches (SWEBOK 2004).

One approach taken to achieve best practice is to use Verification and Validation techniques (Boehm 1984). The aim of validation is to test the outputs of each of the development activities against the requirement specification. Verification is more akin to software quality frameworks, in that it attempts to ensure that a suitable and rigorous development life-cycle is pursued. Testing is a form of validation.

1.2 Aims and objectives of the research project

POSE is a framework for requirements engineering and design, and is in its early stages of research. There has been no research into the relationships between POSE and testing, this dissertation aims to investigate this.

Hall et al. (2007a) introduce the notion of using test cases as an alternative and cheaper form of validation than that offered by justification and assurance based development.
The idea is that by designing tests, these could be validated by the client. Testing the hypothesis that it would be cheaper is outside of the scope of this dissertation. This dissertation aims to do is supply the initially thinking into how POSE can be extended to allow for what I call Test-Based Justification (TBJ).

By taking usability non-functional requirements, as an exemplar to consider and theorise with, I explore how non-functional requirements might be specified in POSE and test cases designed. Non-functional requirements (NFRs) cover a wide variety of qualities that a software product must realise. Qualities range from usability qualities such as learnability, to changeability qualities such as testability, to functionality qualities like security. Specification and testing of these qualities can be a problematic task, due to the breadth and depths of these specialist fields. NFRs are also notoriously more difficult to elicit from clients, than functional requirements are.

The only quality explicitly researched in POSE is safety, which appears to be a unique because it is not specified as such, but all requirement and designs need to be justified or assured to be safe. For the scope of this dissertation it will not be practical to consider more than one type of quality requirement within POSE, due to the depth of knowledge needed. I will consider Usability qualities, having completed the post graduate OU course “User Interface and Design and Evaluation” M873, and having practised the theories professionally.

To support investigation of NFRs I believe it is necessary to first consider how test cases can be designed for functional requirements. I assume functional requirements will be a simpler problem than NFRs, and this preliminary investigation will expose some of the basic techniques that might subsequently apply to NFR test case design.
Using the “Tax Form” (Beizer 1995) case study, I intend to explore and suggest answers to the following questions:

- How test cases for functional requirements can be derived from POSE.
- How NFRs can be specified within POSE and test cases designed. Taking “Usability” NFRs to work through and reason with.
- What are the relationships between test design, POSE transformations, POSE sequents, and the POSE problem tree.
- To identify how test case design may help to validate activities within the POE process pattern.
- How to extend POSE to incorporate test-based justification, accompanied by some comparison of the options.

1.3 Summary

The aim of the project is to test the hypothesis that test-based justification of POSE is possible. The outcome of the project will be to present possible POSE extensions for test-based justification. In order to achieve this I will need an understanding of both POSE, and testing. I will achieve this by application of theory. Subsequently I will consider how test-based justification in POSE.

I make the assumption that software testing concepts are appropriate when considering test-based justification.

Defining and specifying tests will provide an additional form of validation for POSE specifications, and could also improve the quality of the requirements.

Acceptance tests will be needed for functional, non-functional and constraint requirements. NFRs can have cross cutting concerns across a specification, and
therefore need special consideration. Functional requirements can be tested using well
known black-box testing techniques.

POSE specifications have defined aspects, which to extend in any way will need either
change or additions.

The next chapter will take a closer look at the context of this project.
Chapter 2  The Problem in Context

2.1 Introduction

This project will synthesise problem oriented software engineering (POSE), which is new, with older and well understood aspects of software development. This chapter will start with an overview of POSE and then its nouns, verbs and patterns – these elements may need extension to achieve test-based justification (TBJ). TBJ of POSE may impact other areas of software engineering, including requirements engineering, the software development life-cycle, project management, software quality and finally testing, each of these is discussed in this chapter.

2.2 Problem Oriented Software Engineering

POSE is a framework that supports requirements engineering and design. It allows the formal and non-formal can be captured, and recorded side by side. Non-formal domains (the real world) present 2 main difficulties, in the words of Turski:

1. “Properties they enjoy are not necessarily expressible in any single linguistic system.”
2. “The notion of mathematical (logical) proof does not apply to them.”

The aim of POSE (Hall, et al. 2007) is to provide a framework that allows problems to be understood and solved. The understanding equates to requirements, and the solution equates to the specification. The output of POSE is a requirement and design specification that has been justified, and therefore validated. The POSE specification has a hierarchical structure with each node being textually recorded using the justification general form.
2.2.1 POSE Features

POSE allows for exploratory investigation of both the problem and the design, taking the same view as Vincenti that “engineering is a problem solving activity”.

POSE is agnostic of development process. It is suitably open that it is not constrained to work within any particular process. This allows managers to adopt the most appropriate process for a project and still use POSE. However there is limited evidence of this in practice.

POSE progressively gathers requirements and allows the specification to be worked with in an interleaved fashion. The requirements and specification are recorded side by side, thus providing clarity and traceability. In this approach it would also be difficult to omit a solution by human error, because of the side by side nature of solution and requirement. Arguably most importantly, interleaving is appreciative of the fact that design choices may lead to further requirements.

POSE is based on Gentzen sequent calculus. A sequent is a POSE problem and is written as $W, S \vdash R$. This mathematical proof gives POSE its ability to be used with formal methods.

POSE is exploratory, it means that some paths will not form a workable specification, at which point “back-tracking” (Rapanotti et al. 2008) occurs. These failed paths are retained in the problem tree, because a) they provide traceability b) they are likely to result in adding useful information into the specification.

There are several forms of sequent transform, each operates on either S or R, and can also add to W.
The selection of the transforms is based on context. The decision of the appropriate transform may vary between people, as it is a subjective choice based on many things, including experience.

POSE allows for adequacy arguments to be incorporated into the design process. These are in the form of justifications for each sequent, thereby making traceability a trivial matter.

POSE allows the use of modelling notations (Rapanotti et al. 2008) such as Problem Frames and the UML. This allows usage of the most appropriate tool for the task, and does not force a new and needless notation to be learnt.

### 2.2.2 POSE Problem

In POSE the problem is the atomic unit that is manipulated using transforms within a design process pattern. The problem is comprised of three elements:

1. The requirement. A single requirement.
2. The solution to the requirement.
3. The context into which the solution to the problem must fit. The context applies to the requirement as well as the solution. One solution can be part of the context of another.

The problem is written in the form of a mathematical sequent, which comes from proof theory (Wikipedia). A mathematical sequent is presented as $\Omega \vdash \Sigma$, $\Omega$ and $\Sigma$ are sequences of formula, and the $\vdash$ read as either turnstile or tee, means “proves”. $\Omega \vdash \Sigma$ therefore expands to read “$\Omega$ proves $\Sigma$”.


The POSE form of the sequent is $W, S \vdash R$ where $W$ is the world (the context), $S$ is the solution and $R$ is the requirement.

### 2.2.3 POSE Nouns

The $W$ component of $W, S \vdash R$ is a domain, and conforms to the following structure “$D = N : E$” where $N$ is a name and $E$ is a description, both $R$ and $S$ have the same structure. $D$ has an alphabet of three types of phenomena – controlled, observed and unshared.

### 2.2.4 POSE Transforms

Hall and Rapanotti (2008g) groups the possible transforms into 4 categories:

1. Problem Exploration. By using either problem structuring or interpretation.
2. Solution Exploration. By solution structuring or interpretation.
3. Associating Problem and Solution. By problem progression or solution expansion.
4. Structural. Such as specifying architecture.

### 2.2.5 POSE Justification

During the course of writing a POSE specification the justification case is built up and incorporated within the specification. Each transform has a justification of how the transform was reached from the previous problem. A POSE the specification is seen to be justified if all its justifications hold true.

Justification is intended to convince stakeholders that:
1. Their individual concerns are met.

2. The solution meets the requirements in the defined world.

3. The requirements are understood in context of the defined world.

There may be multiple stakeholders, each with differing justification needs.

The mechanism of justification allows multiple cross-cutting concerns such as those required by safety standards, to be included against each transform. The safety standard concerns (taken from Hall, J., Rapanotti, L. 2007c) are:

1. Well founded

2. Reliability

3. Sound Judgement

4. Feasibility

Hall et al. (2008f) use justification descriptions to state one of three things about the transformation:

1. “why we chose this”

2. “what’s unclear”

3. “may need further work”

Hall et al. (2007a) suggests that there an alternative to this form of justification would be test-based justification.
2.2.6 POSE Patterns

As a result of using POSE in practice, Hall et al. (2008) determined that a common pattern emerged for the application of POSE. They found that it was necessary to understand the problem first before determining any solutions; this has become known as the POE Process Pattern, as shown in Figure 1 The POE process pattern.

Figure 1 The POE process pattern

Hall and Rapanotti (2008a) concluded that there are 3 ways the pattern can combine with itself. The first approach is sequential, as used during the Preliminary Safety Analysis work of Hall et.al (2007b). Each problem is solved individually from the top-down, this is shown in Figure 2 Sequential POE Process.
The second approach is parallel, see Figure 3 Parallel POE Process, an example of its use would be how open source projects occur. Many people could be working as
individuals on the same problem, and then the best solution is selected from the entire set.

Figure 3 Parallel POE Process
The last approach is fractal; see *Figure 4 Fractal POE Process*. In Hall and Rapanotti (2008) we see that fractal design considers the two non validating activities of stating the requirement and specification as being exploratory, and are considered as individual problems to be resolved using the POE process pattern. The sequent for problem exploration is:

| Problem Validation Context, Problem Validator, Problem Description | Problem Validation Criteria |

The output from this that feeds into the containing process pattern is the problem description. Problem Validation Context refers to the world, the Problem Validator is the person responsible for validation, and finally the Problem Validation Criteria refers to the requirements of the problem.
The sequent for solution exploration is the same, but replacing problem for solution in the sequent:

\[
\text{Solution Validation Context, Solution Validator, Solution Description} \rightarrow \text{Solution Validation Criteria}
\]

These sequents are shown pictorially in the following diagram:

**Figure 5 Problem and Solution Exploration as Problems**

2.2.7 POSE Specification Examples

Three papers that contain detailed examples of POSE specifications. First Hall et al. (2008f), specify an industrial control system. Second Hall et al. (2008d), which specifies a decoy controller, part of a safety-critical, embedded and military solution; this is the most detailed specification available. Lastly Hall et al. (2008b) which is
very different in that it is capturing knowledge, it produces a model of the publishing industry.

2.2.8 Justification General Form

Each problem is written conformant to a structure known as the “justification general form” (Hall et al. 2008d). The overall specification, the primary output from POSE, is a document containing all of the justification general forms; these are sequenced to show hierarchical order.

The justification general form contains the following sections:

- **Step Identifier** – to uniquely identify the problem.

- **Justification** – an identifier. Though Hall et al. (2008b) it contains narrative.

- **Descriptions and Phenomena** – domains, attributes and associations that are added to the problem during this transform.

- **A concern block** that includes a name, status, claim, argument, evidence, and risks. This relates to assurance based development as used with safety-critical systems.

- **A specialist concern block** for step validity containing status, argument and evidence and signatory. This block is always present and provides a summary of the justification state for the transform.
2.2.9 Related Work

POSE is in the process of being generalised into Problem Oriented Engineering, and from there into Natural Design.

The related work, of most significance to this dissertation, falls into three main areas:

1. Problem Frames – provide a framework for requirements engineering, and are the conceptual basis of POSE.
2. Goal-Oriented Requirements Engineering. In this requirements are written as goals, which can then be decomposed into further goals. The decomposition forms a tree structure; the leaf nodes represent the specification. One form of Goal Oriented RE is KAOS, which is currently being researched with a view to linking it in the subsequent development activities.
3. Relating the POE process pattern to Rational Unified Process (RUP) activities (Kaminski 2008).

2.3 Formal Methods

Formal methods can be utilised to either generate specifications or to verify a system (Sommerville 2001). When used for specifications, the output is provable to be correct; this has made formal methods useful for developing safety-critical systems. Formal methods allow rigorous basis for specification by adopting mathematical notation, examples include Alloy and Z notation.

Mannering et al. (2007) is an example of using POSE with formal methods. This is interesting for my project because it shows POSE used in conjunction with well known methodologies and techniques.
2.4 User Centred Design

User Centred Design (UCD) uses engineering principles to approach user interface design (Preece et al. 1994). Poor user interface design was a contributing factor in the disaster at Three Mile Island in 1979, where a nuclear power plant nearly went into meltdown (Open University 2001). This demonstrates that user good interface design can be critical for safety-critical systems. Moreover, UCD could be central to the success of any project that has a user interface.

UCD is an iterative process spanning requirements, design and evaluation (Open University 2001). The iterations may be repeated as many times as necessary to arrive at a satisfactory solution.

UCD Requirements gathering concentrates on:

1. Context - Determining the domain the system operates in. This refers to high level domains, for an ATM the primary domain would be banking, so users of an ATM are expected to know terminology and activities involved in personal banking.

2. Who - Identifying the users, and their features. Techniques include User Profiling and Personas.

3. What – the tasks that the system enables the users to fulfil. Thinking in terms of Goals, Tasks and Actions. Requirements could be written as Use Cases, Scenarios.

4. Where – the location and the environment the system will be used in. This includes physical, safety, social, organisational, and user support.

The design activity uses work re-engineering, analysis of how users will perform the desired tasks, and conceptual design, before moving on to the physical design of the system.
Evaluation is performed by either observing users or using heuristics. The choice of evaluation and how it will be executed are determined in the evaluation strategy.

UCD has similarities to POSE in that:

1. Both have a requirements gathering/specification and design phases
2. Both allow iteration.
3. Both provide a means of validation, POSE through its justifications, and UCD via evaluations.

2.5 Software Development Life-Cycle

The aim of a software development life-cycle is to mandate the choice and sequence of activities that as a whole will produce software. The Open University (2006a) lists the following activities contained within the software development:

- Requirements Elicitation
- Requirements Specification
- Estimation
- Architectural Design
- Project Planning and Control
- Detailed Design
- Module Implementation and Coding
- System Integration
- System testing
- Acceptance Testing

Each of the activities has deliverables that inputs into subsequent activities. The sequence of activities is defined by the life-cycle model. There are many models, including waterfall, incremental, iterative, evolutionary and Vee-model.
Figure 6 Vee life-cycle model (taken from The Open University 2007) shows the relationship between testing and the other software testing activities, testing levels in Myers (2004) terms. Defining testing levels is valuable because it allows inefficiencies of duplicated effort between testing levels to be removed, and also that all testing levels can be considered as a whole unit.

Understanding the connections between POSE and Testing in the Vee-model will be relevant to TBJ.

2.6 Requirements Engineering

The Open University (2006b) demonstrates the requirements process as an iterative process that includes the following activities:

1. Eliciting Requirements
2. Modelling and Analysing Requirements
3. Communicating Requirements
4. Agreeing Requirements
5. Evolving Requirements
2.6.1 Fit Criteria

Robertson and Robertson (1999) introduce the notion of adding a Fit Criteria to each requirement. A Fit Criteria is a quantifiable benchmark; if the built system meets the benchmark then it proves that the requirement has been met. While Fit Criteria feed into the test design process, they are not in themselves tests. Stating the requirement and its Fit Criteria is iterative process, each enhancing the other until they are both of an acceptable standard. It is possible to retrospectively add Fit Criteria, in which case the requirements are likely to need updating too. The IEEE standard 830-1998 states that a good requirement is verifiable and testable, both qualities Fit Criteria supply.

Sommerville and Sawyer (1997) present a semantically equivalent idea but they confusingly use the term test-case.

Each requirement can be classified as either functional, non-functional or constraint. Fit Criteria can be written against each of these.

2.6.2 Quality Gateway

The quality gateway (Robertson and Robertson 1999) refers to the activities of justifying the requirements and specification to the stakeholders. This entails specification review, which aims include finding missing, ambiguous requirements, to arrive at a cohesive specification.

2.6.3 Functional Requirements

A functional requirement states what the system must do, to allow the user to perform their tasks. Typically they are the drivers for creating the system. For the tax forms
problem, the high level functional requirements might include allowing users to enter tax related information into the system; not delivering this functionality would result in a solution with significantly stunted value. Functional requirements will include the details about the data which the system needs to capture, as well as behavioural aspects.

2.6.4 Non-Functional Requirements

Non-functional requirements are properties or qualities that the system must exhibit. This may include usability requirements such as “a user must be able to complete the tax form process within 20 minutes on their first use of the system”. Other categories of qualities include performance, usability, cultural, maintainability and legal. These can be just as important as functional requirements for the final system. An e-business website will aim ensure that each web page can be served to the user within a given time, because the business knows that users tend to get impatient with online shopping and leave the site, which would have a direct monetary implication. This form of requirement is often harder to elicit from users than functional requirements. This difficulty arises for many reasons, one of which is that users tend to view this as obvious things that do not need stating, however what is obvious from one viewpoint is not always obvious to another.

2.6.5 Constraints

Constraints represent global requirements that restrict the possible solution. They may apply to many aspects of the system such as stakeholders, users, environment, budget and deadlines. Examples of constraints are:
• The system must be web based.

• The system must run on Windows NT4.

2.7 Testing

Bertolino (2007) observes that to develop a working piece of software, testing consumes over 50% of labour resource, and is therefore an expensive activity in comparison to the remainder of the development life cycle.

Bertolino states that testing incorporates many techniques and can be very complex. She presents a clear way of thinking about testing – “Testing always consists of observing a sample of executions and giving a verdict over them”. Her view of testing is that it includes having an objective (e.g. the software is fit to be published), a selection criteria (for the various test techniques), and a notion of adequacy (knowing when to stop).

We are reminded of Dijkstra’s thinking – that testing shows the presence of defects, not their absence. This highlights the fact that if a quality or functionality is not tested for, then it cannot be proven to be met.

Two of the main points regarding the current state of testing research are:

• Identifying test cases is well understood, however their successful combination to form an effective test strategy remains a challenge.

• There is an absence of empirical comparisons of the available testing techniques.

Myers (2004) provides detailed coverage of test case design, and also discusses testing strategy. Myer’s defines testing as “the process of executing a program with
the intent of finding errors”. One way of categorising test techniques is under black-box and white-box.

### 2.7.1 Testing categories

Black-box testing, also known as input/output testing is based on deriving tests on specifications alone. It assumes no knowledge of the internal structure or behaviour. Test data is determined from the specifications. Black-box testing includes the following techniques (Myers 2004):

- Equivalence partitioning
- Boundary-value analysis
- Cause-effect graphs
- Error guessing

White-box testing allows tests to be based on the internals of the program, as well as the specifications. The source of test data is primarily from the internal structure and behaviour, and the specifications.

White-box testing includes these techniques:

- Statement coverage
- Decision coverage
- Condition coverage
- Decision-condition coverage
- Multiple condition coverage

Each technique has strengths and weaknesses – it is capable of uncovering certain types of errors but not others. Therefore using one technique alone will not achieve robust program (Myers 2004). A test-strategy is required; it is the combination of test techniques that are deemed likely to catch the most errors. Each system under test
(SUT) has its own requirements, specification and implementation – hence the test strategy is context dependant on the SUT.

Myers theoretically demonstrates that exhaustive testing is not possible for either black-box or white-box testing. This proof leads into the idea that a testing strategy has a return of investment. A good test strategy will have a strong chance of finding errors.

### 2.7.2 Testing Alternatives

Beizer (1990) highlights that there are alternatives to testing, though he does caveat this by saying the alternatives should be considered as complements to testing, not replacements. These alternatives are:

- Inspections
- Design Style
- Static analysis
- Programming Languages
- Design methodologies
- Development environment

Beizer (1995) based his work on Myers, however Beizer arrived at the following list of black-box testing techniques, which contrast with Myers:

- Control-flow
- Loop
- Data-flow
- Transaction-flow
- Domain
To demonstrate the techniques, he bases all his testing on the US Federal Income Tax Form 1040, and related schedules for 1994. There are no requirements per se, just the forms themselves and the schedules which provide advice on how to fill the forms.

### 2.7.3 Testable Requirements

A difficulty of testing requirements is that the requirements themselves need to have the quality of being testable. If a requirement is unclear or ambiguous, then designing tests will be problematic and of limited value, at least until the requirement is improved. Beizer (1995), Robertsons (1999) and Spillner et al. (2007a) share this opinion, and the importance of having testable requirements.

### 2.7.4 Testing Documentation Standard

The current version of the IEEE documentation standard for functional testing is 829-2008. It is a framework of documents spanning all testing activities. The standard describes how to determine the correct set of test documentation for a given project. The following is a brief overview, of the subset of documents that I will need to employ.

The “Level Test Plan” is a management tool for a particular testing level, such as acceptance testing or system testing.

The “Level Test Design” provides an overview of the testing approaches to be applied at a testing level.

“Level Test Case” is an individual test; it states input and expected output.
“Level Test Procedure” describes in detail how a tester will execute test cases. It lists the series of steps necessary to perform the test.

This standard will be useful for this project, because I will use it to structure and record functional tests.

2.8 Project Management

Project Management is concerned with managing the completion of a project within budget and time to a suitable quality (OU 1995). The main aspects of project management this project is concerned with are the management of risk, resources, cost management, and planning.

2.9 Test Driven Development

Test Driven Development (TDD) spans the area of writing code and unit tests. TDD is an example of a process that inverts a piece of the SDLC, namely coding and unit testing. A benefit of TDD is to encourage the developer to consider how to validate their code using testing, by writing the test before coding solution. TDD is a technique employed within eXtreme programming (xP).

2.10 eXtreme Programming

eXtreme programming (Beck 2000) is an agile methodology. It prescribes an iterative and incremental process which is intended to give customers early visibility of working software.

One of the techniques xP encourages is for the customers to write acceptance tests to validate the software, the developers then write code to pass the test. This idea is an example of test-based justification.
2.11 Software Quality


Software Quality comes from two things:

1. Requirements that state the qualities the system should exhibit.
2. Using best practice techniques in the process of building the system.

Two notable software standards are ISO9001 and CMMI, both are geared towards improving the development process. They have impact across the development lifecycle.

The term software quality management (SQM) includes software quality assurance (SQA), verification and validation (V&V), and finally reviews. SQA and V&V both assess the output from a SDLC stage against the requirements. Boehm (1984) defines verification as “am I building the product right” and validation as “am I building the right product”.

Boehm (1984) shows that the cost of fixing errors gets more expensive the later in the development lifecycle they are found. SQM techniques aim to reveal errors as early in the life-cycle as is sensibly possible.

POSE will produce the requirement specification that will need validating. This research intends to explore possible extensions to POSE to include test-based justification in order to validate the requirements and specification.
2.12 Research question

The hypothesis of this dissertation is that test-based justification of POSE can be achieved through extensions to POSE. The question is how these extensions might look, and what qualities will they have.

2.13 Summary

This chapter has reviewed the existing body of knowledge that is of relevance to the dissertation. The weight of this review has been within POSE, Testing and Requirements Engineering.

The next chapter discusses the research approach taken in this project.
Chapter 3  Research Approach

3.1 Introduction

This chapter will discuss the research methods suitable for this dissertation.

3.2 Research Techniques

POSE is still a relatively new research area, and largely theoretical. The research methodologies I use will need to accommodate for the synthesis of ideas. As part of this synthesis it was necessary to perform a literature review, see Chapter 2, of the following areas:

• POSE
• Requirements Engineering
• Project Management
• Software Development Life Cycle
• User Centred Design
• Testing

The existing POSE specifications comprise of functional requirements. There are no recorded non-functional requirements (NFRs) with the exception of safety. This research will give a fuller picture of POSE and testing, by incorporating tests for FRs,
NFRs and constraints. I will write a POSE specification for a problem which includes a graphical user interface (GUI) because this will provide a source of NFRs and constraints.

There are few POSE specifications and no test cases available them. I need to write a POSE specification and appropriate test cases, to reason with and to develop ideas for POSE extensions.

3.3 POSE and Functional Requirements Testing

I will take the US Tax Forms problem from Beizer (1995), and create a POSE specification for areas that Beizer touches. I will then convert Beizer’s test thinking into test cases written to IEEE 829-2008 standards. On completion I will have:

- gained knowledge of POSE through application
- deeper understanding of functional testing
- experience of writing test documentation
- a POSE specification
- an IEEE compliant set of test cases.

I will have created a platform of knowledge and documentation to base my remaining work on, which I expect to be the more complicated.

3.4 POSE and Non-functional Requirements Testing

Taking the same US Tax Forms, use User Centred Design (UCD) techniques to gather and specify usability requirements, record these in the POSE. The process of UCD is
intended to glean realistic requirements that will be incorporated into the existing specification. My intention is not to suggest how UCD and POSE may be merged into some kind of process or methodology, it is to identify more interesting requirements (particularly NFRs) and add them to the POSE specification.

The motivation for adopting usability NFRs for this project, is that gathering and specifying NFRs is not trivial. It is not unusual to see specialists employed in NFR based fields, such as security, and usability. I have more experience with usability than any other area of NFRs. Bearing in mind that my primary research interest is in POSE, time constraints on the dissertation mean that becoming proficient in an alternative field such as security is just not realistic.

I will not perform UCD on all of the tax forms; one should provide enough material to reason with. I observed that during the Open University’s M873 course that UCD is a time consuming activity and not to be underestimated.

Outcomes:

- A POSE specification with constraints, functional, and NFRs
- Test cases written for NFRs and constraints.

3.5 Test Transforms for POSE

A POSE specification is built as a hierarchy of applied transforms. It seems reasonable that there could be a set of test specific transforms. I will present descriptions for these new transforms.
3.6 Synthesis of Activities

This section forms the crux of the research, and the entailed synthesis is split in two. The inputs into these synthesis activities will be a combination of observations from previous activities, secondary research, and professional experience. The use of professional experience is the weakest of these sources; so I will only apply it in cases where its absence will cause a hole in the research. I have eight years experience as a software developer at a commercial software-house; therefore my experience has some relevance.

The quality of the synthesis is going to be impacted by the quality and range of the inputs. There is the possibility of bias because synthesis is a creative activity, where personal subjectivity can come be an issue.

The first half will be synthesis of testing from a POSE specification, to clarify the relationships between POSE and testing.

The second half is a synthesis to present a description TBJ and its rationale.

3.7 Alternative Approaches

Theoretical research and empirical research are different, and suit different research methods. This project is will use theoretical research methods. There are many research methods for computing, some well described and others less so (Holz et al. 2006).

Formulative Research (Morrison and George 1995) – allows frameworks to be developed and evolved. This sounds very suitable, however there is no detailed description of how to perform it, therefore I ruled this out.
Case Study – the activities in intend to perform for creating the POSE specification and its test designs, might appear to be a case study. However, Easterbrook and Aranda (2006) list four misuses of case study research, of which three can be argued to apply to my research. The three are a) “not be an exemplar” b) “not be an experience report” c) “not be a quasi-experiment”. For these reasons I will not be using a case study as a research method.

Building theory from case-studies (Eisenhardt 1989) – the title is self explanatory, and would present a very strong case for the theory. However there are no existing POSE case-studies incorporating testing, and in the previous paragraph I demonstrated that this dissertation is not a case study. Currently there is not the raw material to use this research method.

3.8 Summary

This research will be based on the “Synthesis” research method.

I will perform the following activities:


2. Record functional test cases for it, based on Beizer.

3. Perform UCD and incorporate the requirements into the POSE specification.

4. Design tests for UCD NFRs and constraints.

5. Present test specific transforms.

6. Synthesise observations from activities to this point, to clarify the relationships between POSE and Testing.
7. Synthesis activities to this point to present test-based justification.

The next chapter will cover the first 5 points of this list.
Chapter 4  Designing POSE, Tests and Testing Transforms

4.1 Introduction

With the secondary research complete, I move onto the start of the primary research. This chapter starts by illustrating the activities performed to generate the POSE specification, which acts as a platform for subsequent chapters.

Having written a POSE specification for the Form 1040 tax forms, the next task was to write some tests based on established testing theory and standards. The specification includes examples of the 3 basic types of requirements functional, non-functional, and constraint. This chapter discusses how the tests were written.

The third part covered in this chapter is a presentation of ideas for recording tests, based on POSE transforms.

The goal of this chapter was to provide raw material that can be used to gain an understanding how testing relates to POSE.

4.2 Writing the POSE specification

4.2.1 The Problem

Beizer (1995) contains detailed thinking about the black-box testing a non-trivial problem. He chose the US individual tax form (Form 1040) for 1994, he presented ways of modelling a test subject in order to identify test cases. He did not attempt to
test the whole of the form, but sections of it that demonstrate a variety of testing strategies.

The first task I undertook was to understand which pieces of Form 1040 he used for which tests. Like Beizer I did not intend to completely work through the Form 1040 problem, my intention was to provide enough material to reason with.

After comparing the 1994 tax forms printed at the back of Beizer(1995) but otherwise unavailable, to the 2007 tax forms (available online) and seeing that they were close enough to cause me no significant problems, I decided to use the 2007 Form 1040. An additional benefit of this is that the IRS also provide a detailed guidance booklet for the form, this would act as an additional source of material for the specification.

The initial question for writing the specification was where to start; I opted to start at the level of Form 1040. The goal of the specification was for it to be as self-contained as possible, at this point it was apparent that without background and context the Form 1040 problem was fairly meaningless. Therefore moved back up the problem tree and inserted the additional necessities.

I used Hall et al. (2007a) and Hall (2008g) as guides for producing the specification. In addition to these, I made use of the documented POSE examples. One thing that became apparent during this process is the creativity involved in POSE.

4.2.2 User Centred Design

The objective for this task was to expand the specification to include non-functional requirements.
I used the process for gathering and recording UCD requirements, as described in OU (2001). The approach was to document:

1. Domain knowledge required.

2. Users, their characteristics and resulting requirements.

3. Users work, including task analysis and resulting requirements, then workflow analysis and its resulting requirements.


After writing a UCD requirements document (see Appendix B - UI Requirements), I updated the POSE specification to include the newly discovered requirements. This included all three types of requirements - constraint, functional and non-functional.

As a part of this merging activity, I split the POSE specification such that model and UI were separated. The resulting specification is summarised in the next section.
4.2.3  POSE specification

The resulting POSE specification is summarised in the following table:

<table>
<thead>
<tr>
<th>Identity</th>
<th>Transformation</th>
<th>Problem Summary</th>
<th>Requirement Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROBLEM EXPLORATION</td>
<td>Problem Overview (IRS, Tax Payer, Authorised Tax Professionals, Online Individual Tax Return - E-Form 1040)</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- web based</td>
<td>constraint</td>
</tr>
<tr>
<td>1.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- include schedules and forms</td>
<td>functional</td>
</tr>
<tr>
<td>1.3</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- scalability</td>
<td>NFR</td>
</tr>
<tr>
<td>1.4</td>
<td>CONTEXT EXPLORATION</td>
<td>- Tax Forms - Components</td>
<td></td>
</tr>
<tr>
<td>1.4.1</td>
<td>CONTEXT EXPLORATION</td>
<td>- Line</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>SOLUTION EXPANSION</td>
<td>- split into model and UI and functional</td>
<td></td>
</tr>
<tr>
<td>1.5.1</td>
<td>PROBLEM PROGRESSION</td>
<td>- MODEL</td>
<td></td>
</tr>
<tr>
<td>1.5.1.1</td>
<td>CONTEXT AND REQUIREMENT EXPLORATION</td>
<td>- - - Individual Tax Forms</td>
<td></td>
</tr>
<tr>
<td>1.5.1.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - only form 1040 in scope</td>
<td></td>
</tr>
<tr>
<td>1.5.1.2.1</td>
<td>CONTEXT INTERPRETATION</td>
<td>- - - Form 1040 - Sections</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.1.2.2</td>
<td>PROBLEM PROGRESSION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.1.2.2.1</td>
<td>CONTEXT AND REQUIREMENT EXPLORATION</td>
<td>- - - - Filing Status.docx</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.1.2.2.2</td>
<td>CONTEXT AND REQUIREMENT EXPLORATION</td>
<td>- - - - Income.docx</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2</td>
<td>PROBLEM PROGRESSION</td>
<td>- - UI</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>1.5.2.1</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - System will not restrict possible user base</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.1.1</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - colour scheme friendly for the colour blind</td>
<td>constraint</td>
</tr>
<tr>
<td>1.5.2.1.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - colour contrast and large font</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.1.3</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - English and Spanish</td>
<td>constraint</td>
</tr>
<tr>
<td>1.5.2.1.4</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - not dependant on audio</td>
<td>constraint</td>
</tr>
<tr>
<td>1.5.2.1.5</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - screen reader friendly</td>
<td>constraint</td>
</tr>
<tr>
<td>1.5.2.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - System will be usable</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.1</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - simple language where possible</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - UI needs to include a help system</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.2.3</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - unambiguous component identity</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.2.4</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - support flexible workflow</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.2.5</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - sub-task status must be clear - complete, incomplete, not started</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.2.2.6</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Number of times user has to refer to a third party for advice</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.7</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Time to commence entering data into the solution</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.8</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - No. of times user referenced docs not included with the solution</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.9</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - No of times the UI misled the user</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.10</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Number of users preferring this solution</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.11</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - % who would recommend to friend</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.12</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Number of times user is frustrated by the system</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.13</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Time to complete Form 1040</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.2.2.14</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - - Time to show new form/screen</td>
<td>NFR</td>
</tr>
<tr>
<td>1.5.3</td>
<td>PROBLEM PROGRESSION</td>
<td>- - FUNCTIONALITY</td>
<td></td>
</tr>
<tr>
<td>1.5.3.1</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - send e-file to IRS</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.3.2</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - store current session</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.3.3</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - retrieve previous session</td>
<td>functional</td>
</tr>
<tr>
<td>1.5.3.4</td>
<td>REQUIREMENT INTERPRETATION</td>
<td>- - upload or identify existing form</td>
<td>functional</td>
</tr>
</tbody>
</table>
I added a requirement type column, which would not normally be explicitly recorded within POSE. The expectation is that this detail will help in identifying tests, and also aid subsequent discussions.

In the problem summary column I have added “-“ to represent each level of depth in the hierarchy.

I did not add concerns to the specification, because there were no real clients, adding concerns would have been too fictitious and misleading.

4.3 From POSE specification to Tests

4.3.1 Testing Functional Requirements

The task was to produce test documentation, based on IEEE 829-2008, for a subset of the tests described in Beizer (1995). Arguably the single most suitable test strategy for Form 1040 is data flow testing, because the problem can be viewed as a calculator.

Conforming to IEEE 829-2008 meant there was a need to produce 4 separate documents:

- Acceptance level test plan
- Acceptance level test design
- Acceptance level test cases
- Acceptance level test procedure
The problem I tested in this piece was:

\[ P_{1.5.1.2.2.2} : \text{Form 1040, Form W-2}^{\text{total}}, \text{Schedule B}^{\text{interest, dividend}}, \text{Schedule C}^{\text{total}}, \text{Schedule C-EZ}^{\text{total}}, \text{Form 4797}^{\text{total}}, \text{Schedule E}^{\text{total}}, \text{Schedule F}^{\text{total}}, \text{Income}^{\text{total FW}}. \]

This problem sits within the Model branch of the specification; however these are acceptance tests and so assume the presence of a user interface. The tests include that the data is enterable, and that the income total is calculated correctly.

The selected tests would also partially exercise, though not enough to validate it:

\[ P_{1.4.1} : \text{Form, Line}^{\text{value reference, look up}} \Rightarrow \text{Line} \]

This problem is a cross-cutting functional requirement, which declares how data-entry is performed.

### 4.3.2 Testing Non-Functional Requirements

The tests covered here fall outside of theories presented in Beizer (1995).

To get a broad sample of NFR tests, I took one usability NFR and one performance NFR.

**Usability Testing**

The usability NFR under test was “The percentage of users who would recommend the solution to a friend will be at least 70%”.

The goal was to provide an outline for testing above requirement. In UCD terms this is actually an evaluation against a prototype of the system or the final system.
Typically evaluations seek to prove or disprove several requirements at once. Therefore if I were to add tests for all usability NFRs we would see that these could be answered in fewer evaluations than there are NFRs. As an example other NFR that could be tested during this evaluation would include:

- “Regarding the usability notion of learnability, users will need to reference documentation not contained within the system, will be less than 1 per 15 users.”
- “Regarding the usability notion of throughput, the number of times a user is frustrated by the system will be 2 times or less per filing.”

UCD works in an iterative manner, of design and evaluation, each feeding into the other. Therefore we would expect this evaluation to be performed more than once, at different times. However, it is possible however that one evaluation will prove the system meets the requirements and needs no further improvement.

**Performance Testing**

The performance NFR under test was “Regarding the notion of performance, the time for the system to show a new web page will be 4 seconds or less.”

For this I wrote an outline for a test that could be performed that would validate the system meets the requirement, see Appendix G – Non-Functional Requirement Tests.

**4.3.3 Testing Constraints**

*Appendix A - POSE Specification* contains the following constraint:

“The customer has stated that the system must be web based.”
The question is how can do test this, using software testing theory. This falls outside of Beizer (1995) theory. The answer is that this constraint is not testable; however it can definitely be validated by inspecting the specification.

There are three more examples of, what are arguably, constraints derived from the UCD:

- *The colour scheme of the UI must be friendly to the colour blind.*
- *The colour scheme must be high contrast to aid the partially sighted.*
- *The UI must be available in both English and Spanish.*

All three of these constraints are testable, with varying degrees of difficulty. Appendix H – Constraint Requirement Tests contains a test for the “English and Spanish” requirement.

### 4.3.4 Test Strategy

To recap on testing theory, before writing designing individual tests, it is necessary to take a more holistic view of the solution and identify the most appropriate test strategy or strategies to apply. This choice directly impacts tests to be designed.

I chose to record the testing in terms of the IEEE Standard 829-2008 for Software Test Documentation. The test strategy I employed is recorded in the “Approach Refinements” in Appendix D - Level Test Design.

Neither the IEEE standard or test strategy theory caters for NFRs or constraints. Therefore they are not recorded in the test strategies section of my IEEE conformant documentation.
## 4.4 Proposed Test Transforms, Nouns and the Sequent

Having written both a POSE specification and tests against it, the next step was to consider how test-based justification may be recorded. This section explores and describes the options that I have considered. The presentation of each option will include its application against a single problem. The problem to test will be:

\[ P_{1.5.1.2.2.2} : \text{Form 1040, Form W-2}^{\text{total}}, \text{Schedule B}^{\text{interest, dividend}}, \text{Schedule C}^{\text{total}}, \text{Schedule C-EZ}^{\text{total}}, \text{Form 4797}^{\text{total}}, \text{Schedule E}^{\text{total}}, \text{Schedule F}^{\text{total}}, \text{Income}^{\text{total FW}.} \]

The tests I have for this are recorded in Appendix D - Level Test Design, Appendix E - Test Cases, Appendix F - Test Procedure. It is these tests that will be “merged” into the POSE specification.

### 4.4.1 Testing Nouns

I previously described what I have coined as POSE nouns W, S and R. I make the assumption that a new noun T, for test, may be useful for subsequent work in this section.

This leads to the following candidate for the new T noun:

\[ T = N : E \quad \text{there is no reason to assume that this would need to be different from already established approach.} \]

The “E” part of the noun needs to record the description of the test.
\( T_{\text{observed phenomena}} \) – This would show that phenomena in other nouns are visible and used as inputs into this test. It is not evident that controlled or unshared phenomena exist in the current POE meaning of them.

**Example**

To re-write the problem sequent in this style results in the following test noun:

\[
T \quad \text{Income!total, FW-2!total, SB!interest, SB!dividend, SC!total, SCEZ!total, F4797?total, SE!total, SF!total}
\]

### 4.4.2 Test Transform

In POSE each new problem is arrived at by applying a transform. I propose that there could be new “test” transforms. These new transforms are based on Beizer (1995) and Myers (2004) testing theory.

There would definitely be a “Test Transform” that would describe, in client friendly terms, test(s) for the problem. This could be added as a sub-problem for \( P_{1.5.1.2.2.2} \) resulting in the following justification general form:

**Example**

\[
P_{1.5.1.2.2.2.1}
\]

STEP ID: Application of TEST IDENTIFICATION to 1.5.1.2.2.2

\[
P_{1.5.1.2.2.2.1} \quad T \quad \text{Income!total, FW-2!total, SB!interest, SB!dividend, SC!total, SCEZ!total, F4797?total, SE!total, SF!total}
\]

├ Form 1040, Form W-2\text{total}, Schedule B\text{interest, dividend}, Schedule C\text{total}, Schedule C-EZ\text{total}, Form 4797\text{total}, Schedule E\text{total}, Schedule F\text{total}, Income\text{total} \_FW-

\[
\]

├ Income Section
JUSTIFICATION J1.5.1.2.2.2.1:

There is a need to specify tests for the parent problem.

TESTS:

1. Show that wages (sourced from Form W-2\textsuperscript{total}) are added to the total\_income field.

2. Show that each of the remaining 14 fields can be individually set.

3. Show that all appropriate fields are added to the total\_income field.

4. Prove that any field that is not part of the total\_income calculation, is not included in the calculation.

4.4.3 Test Strategy Transform

A “Test Strategy Transform” may be valuable. The intention would be to broadly describe the testing approach. It is likely that this would be an ancestor for subsequent “test transforms”. The justification general form has a new Test Strategy section; the text for the contents was copied from “Approach Refinements” in Appendix D - Level Test Design.

Example

STEP ID: Application of TEST STRATEGY IDENTIFICATION to 1.5.1.2.2.2
P1.5.1.2.2.2.1: **TS(Data Flow Testing)** Income!total \[\rightarrow\] Form 1040, Form W-2\text{total}, Schedule B\text{interest, dividend}, Schedule C\text{total}, Schedule C-EZ\text{total}, Form 4797\text{total}, Schedule E\text{total}, Schedule F\text{total}, Income\text{FW-2\total,SB\interest,SB\dividend,SC\total,SCEZ\total,F4797\total,SE\total,SF\total}

\[\rightarrow\] Income Section

**JUSTIFICATION J1.5.1.2.2.2.1:**

There is a need to a test strategy to guide subsequent test identification.

**TEST STRATEGY:**

Data Flow Testing – a black box technique. At risk of oversimplifying the problem under test, the solution is calculator and therefore the correct flow and use of data is critical to correct calculation.

**DESCRIPTIONS AND PHENOMENA:** …………

### 4.4.4 Test Transforms and the Sequent

The meaning of transforms in current POSE, is that they change the problem, and the problem is shown to be different in terms of W,S \[\triangleright\text{R}..\] This would lead to the notion that there is a “T” noun that needs to be included within W,S \[\triangleright\text{R}.\] One option is as follows:

\[T \triangleright W,S \triangleright R\] means that the test proves the solution in the world and therefore the requirement.
I presented both the test transforms using this notation. The main advantage of doing this is it is clear what is tested in terms of data and transform, just by looking at the sequent.

4.5 Evaluation of Evidence

There are three areas to evaluate for this chapter’s evidence:


2. Size of subject.

3. Quality of my application of techniques.

The subject is based on the examples from the IRS Form 1040. This project was not based on a live project with real users or stakeholders, which would have been a more convincing source of evidence. However, the problem is well documented by both the IRS and Beizer (1995). This subject allowed me to use the logical functional test designs from a leader in the testing field, Beizer, thereby increasing the quality of my testing work.

The aim of this chapter’s work was to gain enough evidence to input into the synthesis. The evidence shapes the subsequent synthesis; therefore this evidence can be evaluated, to some degree, by the quality of synthesis in later chapters.

The POSE specification was written to requirements level, and does not include design. A review by my technical advisor confirmed that the specification is adequate for purpose.
Application of both NFR and constraint tests was weaker than the functional requirements because they were not based on Beizer logical designs.

Due to the choice of subject, it was not possible to incorporate users or stakeholders into the UCD work. Therefore the UCD work potentially weaker than the POSE and testing activities. This is acceptable, because I have used UCD as a vehicle to provide more requirements.

4.6 Summary

This chapter has described the process I used to generate the POSE specification. I first wrote up functional requirements based on the tax forms problem. Then I performed UCD requirements as a separate process and document, resulting in a batch of new requirements that included NFRs and constraints, which I merged into the POSE specification.

Next I looked at deriving acceptance tests from the POSE specification. Functional tests based on testing theory were fully designed and then recorded using the IEEE 829-2008 standard. NFRs testing can take different forms; I proposed a usability evaluation, and also load testing for performance. Also I showed that multiple requirements may be validated within one evaluation. The testing of constraints was interesting because I found that not all constraints are testable, and that those that are, were most easily documented not using the IEEE standard. Deciding on the test strategy was a useful tool for determining the set of functional tests. However I could not find any software testing theory that directly presents testing strategies for NFRs and constraints.
Chapter 5  Reviewing POSE and Testing

5.1 Introduction
Following the activities of creating a POSE specification and tests against it, I am able to discuss the relationship of POSE with Testing. This section is not attempting to consider test-based justification.

5.2 Software Development Life Cycle
When using POSE, the requirements elicitation and specification stages of the Vee process life cycle, will always be merged as shown in Figure 7 POSE Vee Process Life Cycle.

*Figure 7 POSE Vee Process Life Cycle*
This can be taken further in cases where an solution architecture is designed, seen in Hall et al.(2007c) and where Solution Expansion transforms are used. Therefore the architectural design step is also absorbed into the POSE design stage, resulting in Figure 8 Architectural POSE Vee Process Life Cycle.

**Figure 8 Architectural POSE Vee Process Life Cycle**

A POSE specification has the ability to incorporate which ever modelling notations are useful. This leads to the real possibility of performing component design within the POSE specification. To date there are no recorded POSE specifications written to this level. The adapted Vee model is shown in Figure 9 Component POSE Vee Process Life Cycle.
The various POSE Vee models still allow for the same verification processes to be performed within the POSE stage, as when they were split out in the original Vee model. This means that roles and responsibilities of those involved remains the same. The difference is the documentation passed to lower levels, the POSE specification replaces the requirements specification and architectural design.

This suggests that all design activity outputs are added to the POSE specification. A concern with this additive quality is in achieving clarity for those downstream, including validators, testers, and the next designer down the Vee model. Each of these roles is primarily concerned with the output of the design of one stage, but not necessarily in isolation. This places a requirement on POSE that has not been necessary to date, the requirement is to record the “level” e.g. component design, against each problem. The impact of this problem increases as the size of specification expands, which occurs either when specifying large or complicated problems and also when the specification passes to the next design stage.
Theoretically it is possible for test designs to be written before the requirement or solution. The result would be test-driven requirements and design.

5.3 Presentation

The POSE specification is a presentation of the overall design. This allows for different designers to work their problems in separate documentation if need be and then later merge them back into the POSE specification. I did this when designing (see Appendix B - UI Requirements) and recording UCD work in my POSE specification (see Appendix A - POSE Specification).

The POSE specification is a presentation of the specification, not the process of determining the specification. This record of transforms, and sequence do imply that POSE is a top-down approach, this is not always true, middle out is achievable.

5.4 Non-Testable Requirements

The first step in test design is confirming that the requirement is testable (Beizer 1995), a quality that good requirements have (IEEE Standard 830-1998 and Robertson, S. and Robertson, J. 1999). In cases where this is not true, the tester would need to refer back to the requirement author for further information. The risk in this backtracking is that the clarification is not recorded in the specification.
5.5 POSE

The fact that POSE specifications are recorded top-down does not mean they are developed in this way. My specification was developed middle out. It is important to remember that the POSE specification is a presentation, not the process.

The transforms recorded are at varying levels of granularity, from the precise in Hall et al. (2008b) and Appendix A - POSE Specification, to the coarse in Hall et al. (2008d). Testing coarse grained specifications suggests the need several test cases per transform.

5.6 Testing

Testing is a creative discipline within the SDLC. Testing is executed against software; however acceptance testing and system testing can both have their tests designed in the absence of an executable system. Acceptance testing is based on the R part of the W,S ├ R sequent, while system testing is based on the S piece of sequent.

5.7 POSE Transforms and Testing

The first question is “can all transforms be tested”, in answer to this, it is already clear that some can be tested because all the tests (in Appendix E - Test Cases, Appendix G – Non-Functional Requirement Tests, and Appendix H – Constraint Requirement Tests) relate to particular problems. Having said that the “Problem Progression” transform, which simplifies and/or rewrites the problem is not testable, its purpose is to provide focus for subsequent transforms. Therefore the answer is no, not all transforms can be tested.
If we were to look at groups of transforms and how they may be tested, is there a relationship here? Figure 6 Vee life-cycle model (taken from The Open University 2007) shows that requirements statement feeds the writing of acceptance tests, and that the requirements specification are used to write system tests. The POSE specification integrates the requirements statement and the requirements specification. Therefore it is possible to say that any transform adding information to the requirement and possibly context, which are akin to the requirements statement, can potentially be used for the design of acceptance tests. Also any transform that adds information to the solution and possibly context, akin to the requirements specification, can potentially be used for designing system tests.

The Vee model idea of level testing, is as applicable to testing NFRs and constraints as it is to functional requirements.

The relationship between testing levels and POSE transforms is shown in Table 2. The reason the table says possible level, is twofold:

1. It is intended to be indicative not prescriptive. When POSE is used within a fractal pattern, requirements are explored as individual problems whose solution is the requirement.

2. There is no evidence of component level design with POSE.
Table 2 the Relationship between POSE transforms and Vee Testing Levels

<table>
<thead>
<tr>
<th>Transform</th>
<th>Possible Level Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Acceptance</td>
</tr>
<tr>
<td>Solution</td>
<td>System</td>
</tr>
<tr>
<td>Solution Expansion (to architecture)</td>
<td>Integration</td>
</tr>
</tbody>
</table>

The table does not consider design down to component level, because currently there are no examples, therefore any conclusions would be unfounded.

5.8 POSE Nouns and Testing

All of the functional test strategies in Beizer (1995) concern data and its movement through the system. In POSE specifications, data movement is described in a domain’s “controlled”, “observed” and “unshared” alphabets.

Any data that is declared may be subject to domain tests. Data which is controlled or observed may potentially be tested using Data-Flow, Transaction-Flow, or any of the other strategies described.

There additional circumstances where the strategies may be applied. Problems can be written at varying levels of granularity, those problems that are coarsely written could easily have testable functionality hidden within. Also in Appendix A - POSE
Specification, there are examples of NFRs and constraints, which are not recorded using POSE nouns.

5.9 POSE and Testing Strategies

The notion of selecting suitable test strategies is directly related to the problem area, whereas transforms concern the process of arriving at a solution. These concepts are non-overlapping; therefore there is no clear mapping from testing strategy to POSE transforms.

5.10 The Testing Role

In practice the people performing the testing role are not always test professionals. I have seen project managers, end users, clients, marketing and support staff all fulfilling this role. Therefore the assumption that all testing will be rigorous or based on testing theory is not correct. Of these testers the approaches applied vary from ad-hoc, use case/scenario, to the more formal test strategy based (Beizer 1995).

There are other examples of varied testers. Some forms of testing, such as usability evaluation, require specialists such as usability experts to perform. Some companies off-shore their testing, this may present difficulties in tester availability. In eXtreme Programming the client writes the tests that prove the functionality is complete, this is in addition to developers who are performing test-driven development.

5.11 Summary

The observed, shared phenomena of the POSE nouns are indicative that functional testing is required. However there is no indicator as to the most suitable testing
strategy. Therefore deriving tests must come down to reviewing the specification, and is dependent on the tester’s expertise.

POSE specifications are no different from other specifications, in that you can design tests from them at the appropriate test level.

Requirements must be testable, to facilitate testing.

There is an option of using test specific transforms and nouns to record testing within TBJ. This is be explored in the next chapter when the context of TBJ is more clearly defined.

The next chapter will synthesise the project to present ideas for TBJ.
Chapter 6  Test-based Justification

6.1 Introduction

POSE and its justification is part of wider context, the intention is to deliver adequate software within time and budget.

*Figure 10 POSE and the bigger picture*

*Figure 10 POSE and the bigger picture* shows there are relationships between POSE, development methodologies, project management and quality assurance. It would be possible to expand this diagram further with additional detail in the circles, but as an overview it contains enough for my purposes. This chapter will discuss test-based
justification in context of this diagram, paying particular attention to POSE and the overlaps with it.

POSE is a framework for requirements and design, to extend this framework implies leaving it open enough to allow creativity while being closed enough to remove process concerns.

### 6.2 Levels of Test-based Justification

The need for justification can come from either quality assurance or project management or other stakeholders. Quality assurance would dictate the need a safety-case, for a safety critical system. Project Management may decide that justification is needed as a means of reducing risk. Whoever is the source of the need, will also dictate the level of detail required in the justification.

I believe there are four levels detail for test-based justification, these are:

- **Level 0** – perform usual POSE, with no notion of tests.
- **Level 1** – add Fit Criteria to each requirement.
- **Level 2** – add an overview of suitable tests to the POSE problems.
- **Level 3** – fully design the tests for the specification.

The three important points to highlight at this stage are:

- These levels are incremental; each one is in addition to the previous.
- Each subsequent level and its increased detail represent:
  - A greater reduction of risk, which is important for project managers.
An increase in the effort spent on the requirements and design stage.

- The justification level impacts how test-based justification is performed and the roles involved.

The following sections describe these levels, and will be used a basis for the remainder of this chapter.

### 6.2.1 Level 0 – Do nothing

The motivation for level 0 is that without any test-based justification, POSE specifications are justifiable for two reasons. Firstly, the justification general form contains a justification section for narrative. Secondly, the tree structure of POSE provides traceability which Hammond et al. (2001) argues is a form of justification.

### 6.2.2 Level 1 – Fit Criteria

The intention of adding Fit Criteria (Robertson and Robertson 1999) is to provide testability of individual requirements, thereby enhancing the ability to validate them. Fit Criteria can be used by the designer, as a tool for self-validating the requirements, before going to the stakeholder for their validation. A designer can specify Fit Criteria in the absence of a tester; however a tester is likely to increase the quality (Robertson and Robertson 1999). The inclusion of Fit Criteria is likely to improve the ability to design user-acceptance tests (Sommerville and Sawyer 1997).

I conclude that Fit Criteria are beneficial to POSE regardless of the justification approach, or even the presence of one.
Fit Criteria could be used to stop the potential for infinite recursion in the fractal POE Process. The fractal process drills into further and further detail, Fit Criteria could be used as the measure to determine if drilling should be stopped, because the detail is adequate.

Not all requirements need Fit Criteria. The POSE specification forms a hierarchical breakdown of a problem, with more detail being added as it descends the hierarchy. The Robertsons’ (1999) view of requirements is that they are not hierarchical, unlike POSE. In the spirit of the Robertsons’, I conclude that they would not advocate duplication of Fit Criteria down the problem hierarchy. An example would be the first problem in Appendix A - POSE Specification, it is essentially an overview, but it does have a requirement.

The Fit Criteria narrative will be added to the justification general form as shown in the following example.

**Example**

```
P_{1.5.1.2.2.2}

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to 1.5.1.2.2.2

P_{1.5.1.2.2.2}: Form 1040, Form W-2^{total}, Schedule B^{interest, dividend}, Schedule C^{total}, Schedule C-EZ^{total}, Form 4797^{total}, Schedule E^{total}, Schedule F^{total}, Income^{total}^{FW-2!total,SB!interest,SB!dividend,SC!total,SCEZ!total,F4797!total,SE!total,SF!total}

Income Section

JUSTIFICATION J1.5.1.2.2.2:
```
There is a need to specify the individuals lines present in the income section of form 1040.

FIT CRITERIA:

1. The solution will correctly draw all information into the Income total calculation, each of these inputs will be displayed on the UI.

2. The solution will correctly add all identified inputs, and no more, to produce the income total.

DESCRIPTIONS AND PHENOMENA: …..

Some requirements have more than one Fit Criteria, as shown in the previous example. Fit Criteria can have more than one test; this problem had four tests designed against it (see Appendix E - Test Cases).

Justifications are applied to every transform; therefore there would be Fit Criteria with every new problem.

6.2.3 Level 2 – Test Overview

At this level the designer, and optionally a tester, describe at a high level the tests that would justify the given problem. These test overviews would be written with the intention of being accessible to the validator. They would provide more detail than the Fit Criteria, but less than a test case. Being a high level means that the test overview author does not need specialist testing expertise, but is still able to convey their intent.

From a designers view the intention of justification is to validate their work. Test designers are aiming to write tests that will reveal defects.
Testers do not test everything (Myers 2004), and therefore do not design tests for everything. I have experienced scenarios where new sections of software were not tested, because developers did not tell the testers what needed testing. Testers have finite budget, time, resources, and varying goals, these aspects will be influenced by quality assurance and project management. Testing theory states complete test coverage is impossible (Myers 2004). The choice of testing strategies is deciding the most cost effective way of revealing issues which are of concern (Beizer 1995).

In terms of justification and testing, it is important to understand where the need for designing specific test cases comes from, because it will impact the roles and activities within test-based justification. There are three theoretical ways this relationship can be realised, as shown in Figure 11 the Justification and Test Overlap.

**Figure 11 the Justification and Test Overlap**

Option 1 is that testing contains all necessary justifications. I have already shown that testing can be constrained by various factors. Therefore I cannot conclude that all the tests needed for justification would naturally fall out of usual test activities.

Option 3 is that all tests borne from justifications would include all those from testing. Looking at requirement P1.5.1.2.2.1 “Filing Status” in Appendix A - POSE
Specification, as a designer I would ask if there is any value in justifying that the domain has five possible values, I would simply ask Validator if they agreed. However a tester would want to test that this requirement has been met. Another example that disproves this option is where project constraints dictate that system test design will not occur until post coding. The resulting situation is that justification will need to incorporate system tests that will not be available until sometime after the justification has been validated. Therefore this option is not realistic.

Option 2 is that tests sourced from justification will overlap those from testing. By elimination of options 1 and 3, this is the correct view to take. This conclusion is important, it means that test overviews will need to be written for tests that would otherwise not have been necessary.

6.2.4 Level 3 – Test Design

This is the most granular level, and represents an integration of requirement specification and solution design with test design. The POSE designer will work together with a tester to produce a set of tests, which as a whole validate the specification.

This level increases the work and therefore increases the cost of reaching a validated specification. While at the same time it decreases the overall risk of building the wrong product.

An advantage of working to this level is that the tests will have been written well before the system is ready for testing. This means that the cost of writing tests is moved to an earlier point in the life cycle.
There is a chance that validation reveals issues with the specification, in which case there is the possibility that some of the testing work will need re-work. There is cost associated with this re-work that would not exist if tests were designed against a completed specification.

If a tester was not available for test design and the solution designer had to fulfil that role then there is additional risk. If the designer was blind to a potential problem while solutioning, they are likely to be blind to it when it comes to designing tests.

There are two activities to consider, designing the justification and validating it; these are described in the next two sections.

**6.3 Designing Justifications**

The following subsections discuss the justification design for each level of TBJ.

**6.3.1 Level 1 – Fit Criteria**

Fit Criteria need to be added to requirements before the solution to a problem is designed. Robertson and Robertson (1999) state that adding Fit Criteria may be deferred, but still must be added before specification review. In waterfall requirements and design, the requirements will have been validated before the design activity takes place. In POSE these activities are integrated into:

```
Requirement → Validate → Solution → Validate
```

When incorporating Fit Criteria the cycle becomes:

```
Requirement → Fit Criteria → Validate → Solution → Validate
```
6.3.2 Level 2 – Test Overview

The solution designer and test designer will work closely; the intention is that they avoid any duplication of effort.

Depending how the work is split, it is reasonable to assume that this approach will impose training needs. Both the tester and the solution designer will need an understanding of each others domain. This training incurs cost and needs planning, which is of concern to project management.

The test designer may want to identify which parts of the specification require TBJ, and then write these tests according to a standard test document format. Finally the test designer would merge the test overviews back into the POSE specification.

If the test designer is writing additional tests that are not required for justification purposes, these would not be included in the POSE specification. The motivation for this is to attempt to keep the specification as precise as possible.

During the development of the specification in Appendix A - POSE Specification. The model was developed before the user interface. In test-based justification if I attempted to justify the model in the absence of a UI, the test cases need be written in such a way as to allow this, which is possible.

6.3.3 Level 3 – Test Design

At this level the tests are designed for the appropriate test level.
There are two options for recording them in the POSE specification, 1) add their
totality or 2) add a summation that contains a reference to external test
documentation. The first option would mean merging the POSE specification with the
contents of Appendices C to H. I do not think this is helpful to stakeholders,
validators, designers or testers, because it dilutes the specification. Therefore the latter
option is preferable; however it does raise issues of traceability which will concern
quality assurance.

The incremental nature of TBJ levels would suggest that the Test Overview, in level
2, must be written before the test design. This is not the case, it would be more helpful
to allow test overviews to be written before or after the test design.

The test designer would need to work closely with the solution designer, this does
suggest that co-location would ease the process.

The test designer would be responsible for recording the tests, according to which
ever documentation standard is appropriate. The company I work for has its own
standard, and I would assume it’s a common thing to have.

It may be necessary for the designer to cross-train in the testing standards used. This
incurs cost and time.

The test designer will definitely need to be comfortable reading POSE specifications;
this may be achieved through training, which incurs time and money.

6.4 Validating

Validation can be performed at levels of problem, group of problems (functional or
otherwise) or the whole specification. In addition to validating the justification of each
requirement, there is also a need to perform a specification review. The specification review, includes finding missing requirements and conflicting requirements (Robertson and Robertson 1999).

**Table 3 Validation & Review Frequency**

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Validate Justification</th>
<th>Specification Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual problem</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>A group of problems</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>Entire specification</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 3 Validation & Review Frequency* shows when validation can be performed. Problems only need to be validated once, and specification reviews can only occur in the context of multiple POSE problems.

The variable frequency of justification validation and review, allows for different development methodologies from waterfall based, to incremental (such as eXtreme Programming) and to iterative (DSDM). Frequency is a concern to project managers who typically organise people to be available. My professional experience of validators’ availability has been varied, from SCRUM projects where validators are always present, to situations where they cannot be pinned down and just drop-in when they can.

Validating individual problems or groups allows for earlier identification of issues, thereby reducing risk.
From a designer’s point of view, problems transforms such as problem reduction may not require immediate validation. If the process stated that each problem must be validated before the next problem was approached, this has the effect of the designer being blocked if the validator were not available. By accepting that validating problem groups is possible, the notion of the designer being blocked from further work until validation is performed decreases. Of course there may be key points where validation is required before proceeding, that is to be expected, but they will not occur because of the process getting in the way.

The last approach of validating the entire specification is one that POSE aims to steer clear of, represents a single step and is therefore a waterfall approach.

6.5 Architecture

In the Vee-model architecture specifications are tested via integration testing, which is intended to show the components behave when brought together. Therefore the design of architecture tests is part of integration testing, and is not performed during UAT or system testing.

Recording the architecture justification in terms of “justification” section of the justification general form, is an alternative approach. This allows some level of justification regardless of the testing levels employed. This is supported by the incremental nature of the justification levels.

One view of architecture justification is that the architecture allows all functional, non-functional and constraint requirements to be met. It could be argued that architecture is justified implicitly by justification of problems that are dependent on it.
I do not think this argument for such a critical design piece, is acceptable to validators.

6.6 Review of Testing Transforms & Nouns as Recording Options

The earlier parts of this chapter expanded on the meaning of test-based justification and its processes. I now draw those pieces together to review the test specific transforms and nouns I presented in Chapter 4.

The effect of using test transforms is to add to the problem tree, and to elevate the test to a level equal to the world, solution and requirements. Neither of these effects is positive. Adding many additional test nodes to the problem tree, will complicate the overall document and dilute the specification. Justification elevation from transform component to actual transform is contrary to current POSE. As a designer I have a preference for my problem and justification to sit side by side, which should also assist validators.

A test only has scope and lifetime of the current problem, in terms of keeping the T with the W,S ⊨R of subsequent problems, therefore T is transient. T will not be visible in the W,S ⊨R of child problems, it will reside in one problem only. This differs from the current application of POSE where only the problem progression transform can simplify the problem.

Non-testing transforms could be children of test transforms. This would allow for the possibility that designing tests may reveal further requirements and associated solutions.

Renaming the Test Strategy Transform to “Test Exploration”, would be more in keeping with current POSE transforms. It could contain test strategy for FRs, and
identification of high level test needs such as a usability evaluation. Renaming Test Transform to “Test Interpretation” naturally follows from this. It would contain test case design.

Testing transforms implies a need for testing nouns and their inclusion in the sequent, to maintain the present POSE approach that every transform affects the sequent.

6.7 Test-Based Justification Artefact

TBJ is incremental and its artefacts are additive to the justification general form. Each TBJ level adds more into the form.

The form will have the following new sections:

- Fit Criteria – containing narrative for one or more fit criteria. This conforms to level one – fit criteria.

- Tests – containing test overviews for all justification tests needed by the problem. It may also contain testing objectives, logical tests or test strategy. This conforms to level two - test overview.

- Test Case References – the Tests section will reference tests written within the appropriate testing documentation. This will provide traceability between design and test. This conforms to level three - test design.

TBJ will make use of the existing step validation concern without any modification, because already solves the problem of managing transform status and sign off.

Use of the “justification” section becomes optional, dependent on the context, and the specification author’s view.
Example

The following is a fully worked example based on the ongoing test subject P1.5.1.2.2.2

\[ P_{1.5.1.2.2.2} \]

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to 1.5.1.2.2.2

\[ P_{1.5.1.2.2.2}: \] Form 1040, Form W-2^{\text{total}}, Schedule B^{\text{interest, dividend}}, \text{Schedule C}^{\text{total}}, \text{Schedule C-EZ}^{\text{total}}, \text{Form 4797}^{\text{total}}, \text{Schedule E}^{\text{total}}, \text{Schedule F}^{\text{total}}, \text{Income}^{\text{total}_{\text{FW}}}.\]

Income Section

JUSTIFICATION J1.5.1.2.2.2:

There is a need to specify the individual lines present in the income section of form 1040.

FIT CRITERIA:

1. The solution will correctly draw all information into the Income total calculation, each of these inputs will be displayed on the UI.

2. The solution will correctly add all identified inputs, and no more, to produce the income total.

TESTS:

1. Show that wages (sourced from Form W-2^{\text{total}}) are added to the total_income field. Test Case reference: TC-Income1
2. Show that each of the remaining 14 fields can be individually set. Test Case reference: TC-IncomeN

3. Show that all appropriate fields are added to the total_income field. Test Case reference: TC-IncomeAllIncluded

4. Prove that any field that is not part of the total_income calculation, is not included in the calculation. Test Case reference: TC-IncomeCorrectExclusion

DESCRIPTIONS AND PHENOMENA: ........

CONCERN: (Step Validity) ....

6.8 Summary

The aim of TBJ is to justify the specification so that the process of building the right system can continue. TBJ cannot include test execution as justification, because it would need the system to be built first.

The need for justification is borne from a combination of risk management and cost. To address these needs I have presented TBJ which has three incremental levels - Fit Criteria, Test Overview and Test Design. Each increment increases cost while decreasing risk of building the wrong product. Fit Criteria ensures that problems are testable. Test Overview provides a brief non-technical outline of testing per problem. Test Design is performed by testers in their own standard documentation standard, and referenced by the Test Overview in the POSE specification.
The three levels of TBJ impose differing resource needs and activities on a project. *Table 4 Involvement of Test and Solution Designers in TBJ* shows which TBJ levels each designer is required to work with.

*Table 4 Involvement of Test and Solution Designers in TBJ*

<table>
<thead>
<tr>
<th>TBJ Level</th>
<th>Test Designer</th>
<th>Solution Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fit Criteria</td>
<td>Optional</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Test Overview</td>
<td>Preferable</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Test Design</td>
<td>Mandatory</td>
<td>Optional</td>
</tr>
</tbody>
</table>

TBJ has not precluded the use of existing justification approaches in POSE. TBJ could be used as a complementary technique to ABJ. This is possible because TBJ fits within the POE process pattern, and only makes additive changes to POSE specifications.

I concluded that the testing transforms presented in the previous chapter would not be necessary for TBJ.

Neither TBJ nor ABJ will help with finding missing or conflicting requirements in specification reviews.

The final chapter follows and provides a conclusion to the project.
Chapter 7  Conclusions

7.1 Introduction

The project recorded in this dissertation, is based on the hypothesis that test-based justification of POSE can be achieved through extensions to POSE.

In order to present ideas for test-based justification (TBJ) it was necessary to identify the relationships between test design, POSE transformations, and POSE sequents.

I conclude that there are no precise relationships that apply globally. However there are indicators such as when a POSE noun phenomenon is being observed, is suggestive that functional testing is required. Also it is possible to say, for simple cases, that a) requirements transforms may lead to acceptance level tests, b) solution transforms lead to system tests, and c) solution expansion leads to integration tests. I answered this question in detail in Chapter 5 Reviewing POSE and Testing.

7.2 Project Evaluation

This project was a synthesis and therefore impacted by the quality of secondary research and data gathering. Quality is subjective, however I can say that these areas did provide plenty of material to theorise with.

The project was successful in that I was able to present process and artefacts to support TBJ. Based on perceived variances in the depth of justification required by quality assurance, I propose an incremental approach to TBJ. There are three levels each representing decreased risk but at increased cost, this is relevant to the hypothesis that TBJ may be cheaper than assurance-based justification. Level one “Fit
Criteria” ensures that the problem is testable. Level two is “Test Overview” whereby an outline to a test is added to the problem. Level three is “Test Design” and represents the most granular level of detail. All three levels fit within the POE process pattern.

I achieved this through the synthesis of POSE and testing theory with the practical activities of writing a POSE specification and its test cases.

I recommend the adoption of Fit Criteria for POSE, regardless of the justification approach because having testable problems will have a positive impact on downstream activities, such as testing and detailed design.

7.3 Project Discussion

This section presents the observations I made during the course of the project.

My specialist advisor was invaluable as both a sounding board and as a guide to the practicalities of POSE. In hindsight I should have involved him earlier in the project, because I would have progressed more quickly.

The IEEE functional test documentation standard was easy to understand and adapt to my needs. Writing test documentation to this standard was trivial.

I originally expected that constraints would not be testable, however some were. I was unable to find guidance on how to write tests for constraints.

UCD was a far richer source of requirements that I had expected, and provided a more rounded specification.
From the POSE literature I assumed that POSE was always performed top down. I spent a long time trying to understand how to start the specification at the top and work down. Eventually gave up and started in the middle on P1.5.1.2. However my specialist advisor stated that they can be performed middle out as well.

In order to achieve a cohesive and flowing set of requirements, I did a lot of reshuffling of the problem hierarchy. This was slow process because of the way I had recorded the ids. Use of a tool like POELog would have helped.

While identifying the requirement type, I had difficulty identifying constraints, e.g. “Spanish and English” could be a functional requirement. I think these grey areas are useful because in the real world of software engineering things are not always easy to classify. It is also a demonstration of the subjective choices designers make.

7.4 Project Limitations

This section discusses limitations of the project.

POSE is a creative process, and the limited examples of POSE specifications for guidance, may decrease the quality of my POSE specification.

My specification may not be deep enough or complete enough. It has been a subjective question on when to stop in the specification.

This is the first specification I have seen that is business application based, to date examples have been safety-critical applications, knowledge capture, or embedded systems (package router). Therefore my thinking on the problem domain has no similar work for guidance.
The specification was not produced as part of a real-life project, it is based on an exemplar.

7.5 Future Research

This project achieved its objectives, but in doing so highlights further potential research topics.

Given that TBJ is possible, means that research to test the hypothesis that it is a cheaper alternative to assurance-based justification is now achievable.

While considering the relationship of POSE and testing, I found it necessary to position POSE in relation to quality assurance, project management and development methodologies. It would be helpful to future POSE researchers if a paper was written that examined this positioning.

This project highlighted concerns of traceability and ownership, regarding testing and also multiple authors of a specification. These concerns need to be explored more fully.

I have highlighted that POSE can be used for component level design, in Vee model terms; there are currently no POSE papers that incorporate this design level. It would be useful for both POSE and TBJ if component level design using POSE was researched.
References


[online],


*Research Methods in Computing: What are they, and how should we teach them?* In:
ACM SIGCSE Bulletin, Volume 38, Issue 4 (December 2006)


IEEE (2004) SWEBOK – Guide to the software engineering body of knowledge,
Institute of Electrical and Electronics Engineers Computer Society.


Index

acceptance testing, 6, 21, 43, 57, 59, 138, 140

architecture, 11, 54, 59, 73

Fit Criteria, 23, 63, 64, 65, 66, 69

IEEE Test Documentation Standard, 33, 43, 46, 86, 138, 140, 144, 148

justification general form, 8, 18, 48, 49, 64, 65, 73, 75

POSE Transforms, 11, 57

Test Transforms, 34, 47, 49, 50, 74

project management, x, 8, 29, 62, 63, 67

quality assurance, x, 4, 30, 62, 63, 67, 71

Requirements

constraints, 1, 24, 33, 34, 36, 46, 52, 58, 60, 81, 102

functional requirements, 1, 5, 6, 24, 32, 58

non-functional requirements, 1, 5, 6, 32, 33, 34, 36, 39, 40, 45, 46, 52, 58, 60, 136

test design, 6, 23, 43, 56, 68, 69, 71, 75, 79, 140

test strategy, 3, 25, 27, 52, 60

testing level, 22, 58, 73

User Centred Design, x, 20, 21, 32, 33, 34, 36, 39, 40, 44, 45, 46, 56, 119, 129, 136, 152

Vee Software Development Life Cycle, 3, 21, 22, 53, 54, 55, 58, 59, 73
Appendix A - POSE Specification

This appendix contains a the POSE specification I produced as a result of the activities *Chapter 4 Designing POSE, Tests and Testing Transforms*.

$P_{null}$

STEP ID: Initial null problem

$P_{null}: \text{null, null} \vdash \text{null}$

JUSTIFICATION: starting point for all POSE specification.

$P_1$

STEP ID: Application of PROBLEM EXPLORATION as a starting point to $P_{null}$.

$P_1: \text{IRS, Tax Payer, Authorised Tax Professionals, Online Individual Tax Return} \vdash \text{E-Form 1040}$

JUSTIFICATION $J1$:

The initial problem exploration led to the context diagram shown below. The problem to be solved is to provide an electronic version of the paper based IRS tax form 1040. The solution will be an online application allowing tax payers to electronically enter and submit their tax returns.
Fit Criteria: The solution allows the tax payer to file Form 1040, including related forms and schedules, electronically to the IRS using the standard e-file format.

DESCRIPTIONS AND PHENOMENA:

New descriptions are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inland Revenue Service</td>
<td>The section of the United States “Department of the Treasury” responsible for implementation of tax policies. This includes determining the correct taxes, receiving taxes, and issuing refunds if necessary.</td>
</tr>
<tr>
<td>Authorised Tax Professionals</td>
<td>Authorised to act as a service on behalf of the IRS. They offer services to the taxpayers. For low income tax payers, the tax professionals are subsidised by the IRS. While the remainder of tax payers pay the professionals directly for the use of their services.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Tax Payer</td>
<td>An individual person who is subject to US taxation.</td>
</tr>
</tbody>
</table>

And related phenomena:

<table>
<thead>
<tr>
<th>Name</th>
<th>Phenomena</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-file</td>
<td>The IRS accepts the tax form 1040 in either paper or electronic format. The electronic format is known as e-file.</td>
</tr>
<tr>
<td>Form 1040 – U.S. Individual Income Tax return</td>
<td>The primary tax form required by the IRS for individual tax payers to file their taxes. This form can be different every year, as it reflects government tax policy.</td>
</tr>
</tbody>
</table>
Depending on the tax payer’s circumstances, this form may be accompanied by additional forms, such as Form 3903 – “Moving Expenses”, or schedules, for example “Schedule B”. In all cases, 1040 records summary information from forms and schedules, in the form of a single monetary amount. The tax payer is required to submit all additional forms and schedules used at the point of sending their tax return to the IRS.

<table>
<thead>
<tr>
<th><strong>Tax Refund/Demand</strong></th>
<th>Issued by the IRS to the tax payer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advice and Assistance</strong></td>
<td>Tax forms can be complicated and potentially ambiguous, therefore the IRS and Tax Professionals provide assistance to tax payers in order to correctly file taxes.</td>
</tr>
</tbody>
</table>

\[ P_{1.1} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1

JUSTIFICATION **J1.1:**
The customer has stated that the system must be web based. This is a requirement constraint on the system. This is motivated by IRS stipulation that the system must be web based.

\( P_{1.2} \)

STEP ID: Application of REQUIREMENT INTERPRETATION to P1

JUSTIFICATION \( J1.2 \):

The system must allow electronic means of entering and submitting forms and schedules related to form 1040. The system needs to support all required data needs of the IRS e-file, and also to provide a complete user process for filing their individual tax return.

\( P_{1.3} \)

STEP ID: Application of REQUIREMENT INTERPRETATION to P1

JUSTIFICATION \( J1.3 \):

As stated 100 million Americans electronically filed tax returns in 2006. This activity is performed in a 3 month window each year. The system must be scalable enough to support 50 million users, and still have acceptable user based performance timings (will be stated in a later problem). This is a non-functional requirement about scalability.
**P_{1.4}**

STEP ID: Application of CONTEXT EXPLORATION a generalisation of tax form composition to P1

\[P_{1.4}: AF(s)^{\text{total}}, SCT(s)^{\text{total}}, \text{Tax Form}_{AFs^{\text{total}}SCT^{\text{total}}} \rightarrow E-\text{Form} 1040\]

JUSTIFICATION J2:

This section provides an overview of the contents of tax forms in the general.

![UML Class Diagram](image_url)

**Figure 13 Generalised form composition - a UML Class Diagram**

DESCRIPTIONS AND PHENOMENA

New descriptions are:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCT</td>
<td>Section</td>
<td>A logical grouping of lines.</td>
</tr>
<tr>
<td>LN</td>
<td>Line</td>
<td>Have a textual description, an identifying number and a</td>
</tr>
</tbody>
</table>
value, the latter is supplied by the tax payer.

| AF  | Associated Form | An additional tax form that is required in order to complete the subject form. The outcome of a sub form is either:
|     |                | Not applicable. Often forms start with a series of questions, and/or a flow chart to determine if the tax payer should complete and submit the remainder of the form.
|     |                | A value, typically a calculation based on the sub form lines.
|     |                | It may be necessary to complete this form as part of the process, or alternatively it may be a form that has already been filed with the IRS. |

| SDL | Schedule       | Is associated with a single tax form. May contain one or more worksheets, each of which will yield either:
|     |                | Not applicable. As per sub-form.
|     |                | A value, typically a calculation based on the schedule lines.
|     |                | The overall outcome of the schedule will also be the either n/a or a value. |

And related phenomena:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Phenomena</th>
</tr>
</thead>
</table>

Both associated forms and schedules have a final total that is pulled into the form 1040.

\[ P_{1.4.1} \]

STEP ID : Application of CONTEXT EXPLORATION to P1.4

\[ P_{1.4.1} : \text{Form}, \text{Line}^{\text{value}}_{\text{reference, look up}} \rightarrow \text{Line} \]

JUSTIFICATION J1.4.1:

Form 1040, along with its related schedules and associated forms, share a common form of data entry – the line.

Each line is numbered clearly in bold on the form. Not surprisingly the supporting documentation makes heavy use of these line numbers, for referencing.

Most commonly a line refers to a monetary value. Monetary value entered into the line can be sourced from 4 places:

1. **New** information entered by the user, such as alimony received. For financial data this is subsequently referenced or used in a calculation.

2. A **reference** a previously determined value/line. Either in the same section or a schedule/form. E.g. the wages entered in the income section is supplied by Form W-2

3. A deterministic **calculation**. E.g. add all income entries to yield total income.

4. A **look up** value from IRS supplied information. As is the case in tax rates.
FIT CRITERIA:

1. For New – that the correct data type is enterable.
2. For Reference – that the correct value is used.
3. For Calculation – that the correct calculation is applied, using the correct fields and is rounded correctly.
4. For Look up – that the correct value is used.
5. For all the above – that the value recorded and transmitted correctly.

DESCRIPTIONS AND PHENOMENA:

There are no new descriptions or phenomena.

\[ P_{1.5} \]

STEP ID: Application of SOLUTION EXPANSION to P1

P1.5: Model, UI, Functionality, Individual Tax Process \( \uparrow \) E-form 1040

JUSTIFICATION J1.5: The exploration and specification of the system will be separated into 3 parts:

1. (Domain) Model
2. User Interface
3. Functionality

This separation will provide greater clarity than if it was not performed.
\( P_{1.5.1} \)

STEP ID: Application of PROBLEM PROGRESSION to P1.5

\( P_{1.5.1} \): Individual Tax Process, Model \( \triangleright E\text{-form 1040} \)

JUSTIFICATION \( J1.5.1: \) This step simplifies the problem by removing domains not required to solve it. It allows concentration on the Model.

\( P_{1.5.1.1} \)

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to 1.5.1

\( P_{1.5.1.1} \): IRS e-file, Individual Tax Payer, Form 1040 ES e-file, Form 1040 NR e-file, Form 1040 e-file \( \triangleright E\text{-Form 1040 e-file} \)

JUSTIFICATION \( J1.5.1.1: \)

There are over 300 million Americans, 200 million filed tax returns in 2006, of which over 100 million filed electronically.

The principal form that tax payers need to fill is Form 1040 – Individual Income Tax Return. This form comes in 3 flavours:

1. Form 1040 – Individual Tax Return
2. Form 1040 ES - Estimated Tax Return for Individuals
3. Form 1040 NR – U.S. Nonresident Alien Income Tax Return
These forms are exclusive; therefore a tax payer will submit at most one of these forms.

Tax payers are legally obliged to file tax forms. Each of these 3 forms has a go/no go principle whereby early on in the process it determines if the tax payer is required to file the tax return.

By using the forms the tax payer determines their correct level of taxation for the year, and pays the IRS accordingly. On form completion the tax payer submits the tax form plus any associated documentation to the IRS, and may also include tax payment.

DESCRIPTIONS AND PHENOMENA

New descriptions are:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Form 1040 - Individual Tax Return</td>
<td>IRS tax form for resident tax payers.</td>
</tr>
<tr>
<td>F1ES</td>
<td>Form 1040 ES - Estimated Tax Return for Individuals</td>
<td>Out of scope – due to project constraints.</td>
</tr>
<tr>
<td>F1NR</td>
<td>Form 1040 NR – U.S. Nonresident Alien Income Tax Return</td>
<td>Out of scope – due to project constraints</td>
</tr>
</tbody>
</table>

And related phenomena:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Phenomena</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-file</td>
<td>Controlled by</td>
<td>Both associated forms and schedules have a final total that is pulled into</td>
</tr>
<tr>
<td></td>
<td>F1,F1ES,F1NR</td>
<td>the form 1040.</td>
</tr>
<tr>
<td>Tax Total</td>
<td>Controlled by</td>
<td>The overall tax payment from the tax payer or refund due to the tax payer</td>
</tr>
<tr>
<td></td>
<td>F1,F1ES,F1NR</td>
<td></td>
</tr>
</tbody>
</table>

\[P_{1.5.1.2}\]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.1

\[P_{1.5.1.2}:\text{ IRS}_{e-file},\text{ Individual Tax Payer, Form 1040}_{e-file} \models E-Form 1040_{e-file}\]

JUSTIFICATION \(J1.5.1.2\): The client has stated that only the (standard) Form 1040 is in scope. Therefore Form 1040 ES, Form 1040 NR are out of scope. And removed from the domain

\[P_{1.5.1.2.1}\]

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to P1.5.1.2

\[P_{1.5.1.2.1}:\text{ label, SSN, filing status, exemptions, income, adjusted gross income, tax and credits, other taxes, payments, refund, amount owed, 3rd party, signature, paid preparer, Form 1040} \models E-Form 1040\]
JUSTIFICATION  J 1.5.1.2.1:

Form 1040 is broken into sections; this step identifies and describes them.

1.  Label
2.  Social Security Number(s)
3.  Filing Status
4.  Exemptions
5.  Income
6.  Adjusted Gross Income
7.  Tax and Credits
8.  Other Taxes
9.  Payments
10. Refund
11. Amount you owe
12. Third Party Designee
13. Signatures
14. Paid Preparer’s Section

FIT CRITERIA:

1.  That the sections maintain their descriptive names on the User Interface.
2.  That all sections are present, or represented in an appropriate way. For the later I am thinking in terms of signature (item13), with electronic communications there are alternatives to hand written signatures.

DESCRIPTIONS AND PHENOMENA

New descriptions are:
<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Name and address for postage purposes. May also be used for communication purposes from IRS to tax payer, or authorised tax professional.</td>
</tr>
<tr>
<td>Social Security Number(s)</td>
<td>Uniquely identifies tax payer(s) to the IRS</td>
</tr>
<tr>
<td>Filing Status</td>
<td>One of five options:</td>
</tr>
<tr>
<td></td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>Married filing jointly</td>
</tr>
<tr>
<td></td>
<td>Married filing separately</td>
</tr>
<tr>
<td></td>
<td>Head of household</td>
</tr>
<tr>
<td></td>
<td>Qualifying widow(er)</td>
</tr>
<tr>
<td></td>
<td>This classification affects subsequent calculations.</td>
</tr>
<tr>
<td>Exemptions</td>
<td>Can be tax payer, spouse and/or dependants. Each dependant listed reduces the tax bill by a fixed amount.</td>
</tr>
<tr>
<td>Income</td>
<td>Includes earned (e.g. wages) and unearned (e.g. alimony)</td>
</tr>
<tr>
<td>Adjusted Gross Income</td>
<td>Items that are not taxable (in this calculation), and get subtracted from the current total income. Includes various expenses and education based costs.</td>
</tr>
<tr>
<td>Tax and Credits</td>
<td>Tax this is determined by the tax payer using either a series of tables and/or worksheets, depending on circumstances. Credits reduce the tax bill, and each form of credit has an associated Form or Schedule. There is a child tax credit for example.</td>
</tr>
<tr>
<td>Other Taxes</td>
<td>Applicable taxes not included in the Tax and Credits section.</td>
</tr>
<tr>
<td>Payments</td>
<td>Payments already made to the IRS this year.</td>
</tr>
<tr>
<td>Refund</td>
<td>If Payments exceed the total tax, the tax payer can optionally request a refund.</td>
</tr>
<tr>
<td>Amount you owe</td>
<td>If total tax exceeds Payments</td>
</tr>
<tr>
<td>Third Party Designee</td>
<td>Allow a third party to discuss tax payer’s details directly with the IRS</td>
</tr>
<tr>
<td>Signatures</td>
<td>Of the tax payer, and possibly spouse. A legal declaration of completeness and correctness.</td>
</tr>
<tr>
<td>Paid Preparer’s Section</td>
<td>Details of authorised tax professional, if their services were used.</td>
</tr>
</tbody>
</table>
STEP ID: Application of PROBLEM PROGRESSION to 1.5.1.2

JUSTIFICATION J1.5.1.2.2: Each direct child problems of this one, will consider just one domain and remove the other child domains of Form 1040. To clarify this, the next section will contain Filing Status, but remove label, SSN, Income, etc.

This is intended to help clarify the subsequent problems.

\[ P_{1.5.1.2.2.1} \]

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to 1.5.1.2.2

\[ P_{1.5.1.2.2.1}: \text{Form 1040}_{\text{FS/value}}, \text{Filing Status}_{\text{value}} \rightarrow \text{Filing Status} \]

JUSTIFICATION J1.5.1.2.2.1:

Filing Status is applicable to the tax payer, and each person has at least one status that may apply. The filing status impacts the tax bill calculation. In cases where there are multiple applicable states, the payer should choose the one that yields the lowest tax bill.

There are five possible states:

1. Single
2. Married filing jointly
3. Married filing separately
4. Head of household
5. Qualifying widow(er) with dependant child
These apparently simple values are complicated by scenarios such as divorce, remarrying, married and living apart, death, and contributions to home upkeep. There are criteria that determine the correct value, documented in the 1040 guidance booklet.

DESCRIPTIONS AND PHENOMENA:

New descriptions are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description (copied from 1040 instructions booklet for 2007)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Applicable if any of the following was true on December 31, 2007.</td>
</tr>
<tr>
<td></td>
<td>You were never married.</td>
</tr>
<tr>
<td></td>
<td>You were legally separated, according to your state law, under a decree of divorce or separate maintenance.</td>
</tr>
<tr>
<td></td>
<td>You were widowed before January 1, 2007, and did not remarry before the end of 2007. But if you have a dependent child, you may be able to use the qualifying widow(er) filing status.</td>
</tr>
<tr>
<td>Married filing jointly</td>
<td>Applicable if any of the following apply.</td>
</tr>
<tr>
<td></td>
<td>You were married at the end of 2007, even if you did</td>
</tr>
<tr>
<td>Marital Status</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Married filing separately</td>
<td>Applicable if you are married and file a separate return, you will usually pay more tax than if you use another filing status for which you qualify. Also, if you file a separate return, you cannot take the student loan interest deduction, the tuition and fees deduction, the education credits, or the earned income credit. You also cannot take the standard deduction if your spouse itemizes deductions.</td>
</tr>
<tr>
<td>Not married (Spouse died in 2007 and did not remarry)</td>
<td>not live with your spouse at the end of 2007. Your spouse died in 2007 and you did not remarry in 2007. You were married at the end of 2007, and your spouse died in 2008 before filing a 2007 return. For federal tax purposes, a marriage means only a legal union between a man and a woman as husband and wife. A husband and wife filing jointly report their combined income and deduct their combined allowable expenses on one return. They can file a joint return even if only one had income or if they did not live together all year. However, both persons must sign the return. Once you file a joint return, you cannot choose to file separate returns for that year after the due date of the return.</td>
</tr>
</tbody>
</table>

Married filing separately

Applicable if you are married and file a separate return, you will usually pay more tax than if you use another filing status for which you qualify. Also, if you file a separate return, you cannot take the student loan interest deduction, the tuition and fees deduction, the education credits, or the earned income credit. You also cannot take the standard deduction if your spouse itemizes deductions.
Generally, you report only your own income, exemptions, deductions, and credits.

<table>
<thead>
<tr>
<th>Head of Household</th>
</tr>
</thead>
</table>
| This filing status is for unmarried individuals who provide a home for certain other persons. (Some married persons who live apart are considered unmarried. See *Married persons who live apart* on this page. If you are married to a nonresident alien, you may also be considered unmarried. See *Nonresident alien spouse* on this page.) You can check the box on line 4 only if you were unmarried or legally separated (according to your state law) under a decree of divorce or separate maintenance at the end of 2007 and either *Test 1* or *Test 2* below applies.  

*Test 1*. You paid over half the cost of keeping up a home that was the main home for all of 2007 of your parent whom you can claim as a dependent, except under a multiple support agreement (see page 17). Your parent did not have to live with you.  

*Test 2*. You paid over half the cost of keeping up a home in which you lived and in which one of the following also lived for more than half of the year (if half or less, see *Exception to time lived with you* on this page).
Any person whom you can claim as a dependent. But do not include:

Your qualifying child (as defined in Step 1 on page 15) whom you claim as your dependent based on the rule for *Children of divorced or separated parents* that begins on page 16,

Any person who is your dependent only because he or she lived with you for all of 2007, or

Any person you claimed as a dependent under a multiple support agreement. See page 17.

Your unmarried qualifying child who is not your dependent.

Your married qualifying child who is not your dependent only because you can be claimed as a dependent on someone else's 2007 return.

Your child who is neither your dependent nor your qualifying child because of the rule for *Children of divorced or separated parents* that begins on page 16.

If the child is not your dependent, enter the child's name on line 4. If you do not enter the name, it will take us
longer to process your return.

Dependent. To find out if someone is your dependent, see the instructions for line 6c that begin on page 15.

Exception to time lived with you. Temporary absences for special circumstances, such as for school, vacation, medical care, military service, and detention in a juvenile facility, count as time lived in the home. If the person for whom you kept up a home was born or died in 2007, you can still file as head of household as long as the home was that person's main home for the part of the year he or she was alive. Also see *Kidnapped child* on page 17, if applicable.

Keeping up a home. To find out what is included in the cost of keeping up a home, see Pub. 501.

If you used payments you received under Temporary Assistance for Needy Families (TANF) or other public assistance programs to pay part of the cost of keeping up your home, you cannot count them as money you paid. However, you must include them in the total cost of keeping up your home to figure if you paid over half the cost.

Married persons who live apart. Even if you were not
| **divorced or legally separated at the end of 2007, you are** |
| **considered unmarried if all of the following apply.** |
| **You lived apart from your spouse for the last 6 months of 2007. Temporary absences for special circumstances, such as for business, medical care, school, or military service, count as time lived in the home.** |
| **You file a separate return from your spouse.** |
| **You paid over half the cost of keeping up your home for 2007.** |
| **Your home was the main home of your child, stepchild, or foster child for more than half of 2007 (if half or less, see Exception to time lived with you above).** |
| **You can claim this child as your dependent or could claim the child except that the child’s other parent can claim him or her under the rule for Children of divorced or separated parents that begins on page 16.** |
| **Adopted child. An adopted child is always treated as your own child. An adopted child includes a child lawfully placed with you for legal adoption.** |
| **Foster child. A foster child is any child placed with you by an authorized placement agency or by judgment, decree, or other order of any court of competent** |
| Nonresident alien spouse. You are considered unmarried for head of household filing status if your spouse was a nonresident alien at any time during the year and you do not choose to treat him or her as a resident alien. To claim head of household filing status, you must also meet **Test 1** or **Test 2** on this page. |
| Qualifying widow(er) with dependant child | You can check the box on line 5 and use joint return tax rates for 2007 if all of the following apply. |
| | Your spouse died in 2005 or 2006 and you did not remarry before the end of 2007. |
| | You have a child or stepchild whom you claim as a dependent. This does not include a foster child. |
| | This child lived in your home for all of 2007. If the child did not live with you for the required time, see **Exception to time lived with you** on this page. |
| | You paid over half the cost of keeping up your home. |
| | You could have filed a joint return with your spouse the year he or she died, even if you did not actually do so. |
| | If your spouse died in 2007, you cannot file as qualifying widow(er) with dependent child. Instead, see |
the instructions for line 2 on page 13.

Adopted child. An adopted child is always treated as your own child. An adopted child includes a child lawfully placed with you for legal adoption.

Dependent. To find out if someone is your dependent, see the instructions for line 6c that begin on page 15.

Exception to time lived with you. Temporary absences for special circumstances, such as for school, vacation, medical care, military service, and detention in a juvenile facility, count as time lived in the home. A child is considered to have lived with you for all of 2007 if the child was born or died in 2007 and your home was the child's home for the entire time he or she was alive. Also see Kidnapped child on page 17, if applicable.

Keeping up a home. To find out what is included in the cost of keeping up a home, see Pub. 501.

If you used payments you received under Temporary Assistance for Needy Families (TANF) or other public assistance programs to pay part of the cost of keeping up your home, you cannot count them as money you paid. However, you must include them in the total cost
of keeping up your home to figure if you paid over half the cost.

And related phenomena:

<table>
<thead>
<tr>
<th>Name</th>
<th>Phenomena</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>One of the 5 possible values detailed above.</td>
</tr>
</tbody>
</table>

\[ P_{1.5.1.2.2.2} \]

STEP ID: Application of CONTEXT AND REQUIREMENT EXPLORATION to 1.5.1.2.2

\[ P_{1.5.1.2.2.2}: \] Form 1040, Form W-2\textsuperscript{total}, Schedule B\textsuperscript{interest, dividend}, Schedule C\textsuperscript{total}, Schedule C-EZ\textsuperscript{total}, Form 4797\textsuperscript{total}, Schedule E\textsuperscript{total}, Schedule F\textsuperscript{total}, Income\textsuperscript{total}_\text{FW,-2}\textsuperscript{total,SB\textsuperscript{interest, SB\textsuperscript{dividend}}, SC\textsuperscript{total}, SCEZ\textsuperscript{total}, F4797\textsuperscript{total, SE\textsuperscript{total, SF\textsuperscript{total, FW}}} |

\[ \Downarrow \text{Income Section} \]

JUSTIFICATION \[ 1.5.1.2.2.2: \]

There is a need to specify the individuals lines present in the income section of form 1040.

DESCRIPTIONS AND PHENOMENA

New descriptions are:
<table>
<thead>
<tr>
<th>Line (id)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Wages, salaries, tips, etc.</td>
<td>Amount in dollars. From form W-2. Include spouse’s income if filing status is a joint return.</td>
</tr>
<tr>
<td>8a</td>
<td>Taxable interest</td>
<td>Amount in dollars. From Schedule B</td>
</tr>
<tr>
<td>8b</td>
<td>Tax exempt interest</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>9a</td>
<td>Ordinary Dividends</td>
<td>Amount in dollars. From Schedule B</td>
</tr>
<tr>
<td>9b</td>
<td>Qualified dividends</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>10</td>
<td>Taxable refunds, credits, or offsets of state and local income taxes</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>11</td>
<td>Alimony received</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>12</td>
<td>Business income (or loss)</td>
<td>Amount in dollars. From Schedule C or C-EZ</td>
</tr>
<tr>
<td>13</td>
<td>Capital gain (or loss)</td>
<td>Amount in dollars. From Schedule D</td>
</tr>
<tr>
<td>14</td>
<td>Other gains (or losses)</td>
<td>Amount in dollars. From form 4797</td>
</tr>
<tr>
<td>15a</td>
<td>IRA distributions</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>15b</td>
<td>IRA distributions</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
<td>Amount Information</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>16a</td>
<td>Pensions and annuities taxable amount</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>16b</td>
<td>Pensions and annuities taxable amount</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>17</td>
<td>Rental real estate, royalties, partnerships, S corporations, trusts, etc.</td>
<td>Amount in dollars. From Schedule E</td>
</tr>
<tr>
<td>18</td>
<td>Farm income (or loss)</td>
<td>Amount in dollars. From Schedule F</td>
</tr>
<tr>
<td>19</td>
<td>Unemployment compensation</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>20a</td>
<td>Social Security benefits</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>20b</td>
<td>Social Security benefits taxable amount</td>
<td>Amount in dollars.</td>
</tr>
<tr>
<td>21</td>
<td>Other income – list type and amount</td>
<td>Type is text, amount is in dollars.</td>
</tr>
<tr>
<td>22</td>
<td>Total Income</td>
<td>Amount in dollars. The sum of lines 7,8a,9a,10,11,12,13,14,15b,16b,17,18,19,20b, 21</td>
</tr>
</tbody>
</table>
STEP ID: Application of PROBLEM PROGRESSION to P1.5

P₁.₅.₂: Individual Tax Process, UI ⊨ E-form 1040

JUSTIFICATION J1.₅.₂: This step simplifies the problem by removing domains not required to solve it. It allows concentration on the User Interface.

NOTE: all problems that are children of this one, were originally sourced from User Centred Design activities (see Appendix: UCD-Requirements).

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2

P₁.₅.₂.₁: IRS, User, Individual Tax Process, UI ⊨ E-form 1040

JUSTIFICATION J1.₅.₂.₁: The system will not restrict the potential user base. The client is a commercial concern and wants to ensure that it maximises its potential revenue.

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.1

P₁.₅.₂.₁.₁: IRS, User, Individual Tax Process, UI ⊨ E-form 1040
JUSTIFICATION J1.5.2.1.1: The colour scheme of the UI must be friendly to the colour blind.

\[ P_{1.5.2.1.2} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.1

P_{1.5.2.1.2}: IRS, User, Individual Tax Process, UI \models E-form 1040

JUSTIFICATION J1.5.2.1.2: The colour scheme must be high contrast to aid the partially sighted.

\[ P_{1.5.2.1.3} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.1

P_{1.5.2.1.3}: IRS, User, Individual Tax Process, UI \models E-form 1040

JUSTIFICATION J1.5.2.1.3: The UI must be available in both English and Spanish. Assumes client is happy that this will capture enough of the user base.

\[ P_{1.5.2.1.4} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.1

P_{1.5.2.1.4}: IRS, User, Individual Tax Process, UI \models E-form 1040
JUSTIFICATION J1.5.2.1.4: The UI must not be audio dependent, in order to allow deaf users. The system can use audio, but must provide alternative means of user feedback in addition.

\[ P_{1.5.2.1.5} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.1

P\(_{1.5.2.1.5}\): IRS, User, Individual Tax Process, UI \(\vdash\) E-form 1040

JUSTIFICATION J1.5.2.1.5: The UI must be screen reader friendly to allow the blind users access to the system.

\[ P_{1.5.2.2} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2

P\(_{1.5.2.2}\): IRS, User, Individual Tax Process, UI \(\vdash\) E-form 1040

JUSTIFICATION J1.5.2.2: The UI must be usable. This will be expanded on in child problems.

\[ P_{1.5.2.2.1} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P\(_{1.5.2.2.1}\): IRS, User, Individual Tax Process, UI \(\vdash\) E-form 1040
JUSTIFICATION J1.5.2.2.1: The UI must use simple language where possible. This is a non-functional requirement.

\[ P_{1.5.2.2.2} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

\[ P_{1.5.2.2.2}: IRS, User, Individual Tax Process, UI, Help \vdash E-form 1040 \]

JUSTIFICATION J1.5.2.2.2: The UI needs to include a help system. The help system must, at very least, contain all information in the IRS guidance booklets, for all forms and schedules included in the system.

\[ P_{1.5.2.2.3} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

\[ P_{1.5.2.2.3}: IRS, User, Individual Tax Process, UI, Help \vdash E-form 1040 \]

JUSTIFICATION J1.5.2.2.3: The UI needs to ensure the forms, schedules and their components are clearly and uniquely identifiable. Users may be supported from people who have no knowledge of this solution. The UI needs to make it clear which form/schedule the user is working on, which section, and also maintain existing line numbers.

\[ P_{1.5.2.2.4} \]
STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.4}: IRS, User, Individual Tax Process, UI \rightarrow E-form 1040

JUSTIFICATION $J_{1.5.2.4}$: The UI needs allow users a flexible workflow through the process.

\[ P_{1.5.2.5} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.5}: IRS, User, Individual Tax Process, UI \rightarrow E-form 1040

JUSTIFICATION $J_{1.5.2.5}$: The UI needs to show the status of sub-tasks such as schedules and associated forms. The states could be complete, incomplete, or not started.

\[ P_{1.5.2.6} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.6}: IRS, User, Individual Tax Process, UI \rightarrow E-form 1040

JUSTIFICATION $J_{1.5.2.6}$: Regarding the usability notion of learnability, number of times user has to refer to the IRS or a tax professional for advice, should be one enquiry or less for every 10 users.
**JUSTIFICATION J1.5.2.2.7:** Regarding the notion of availability, users will need to be able to access the system 24/7 for 178 concurrent days per year.

**JUSTIFICATION J1.5.2.2.8:** Regarding the usability notion of learnability, users will need to reference documentation not contained within the system, will be less than 1 per 15 users.

**JUSTIFICATION J1.5.2.2.9:** Regarding the usability notion of throughput, the number of times the system misleads user will be no more than twice per filing.
**P_{1.5.2.2.10}**

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.2.10}: IRS, User, Individual Tax Process, UI \[\rightarrow\] E-form 1040

JUSTIFICATION J1.5.2.2.10: Regarding the usability notion of attitude, the percentage of users preferring this solution compared to previously used solutions will be at least 40%.

**P_{1.5.2.2.11}**

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.2.11}: IRS, User, Individual Tax Process, UI \[\rightarrow\] E-form 1040

JUSTIFICATION J1.5.2.2.11: Regarding the usability notion of attitude, the percentage of users who would recommend the solution to a friend will be at least 70%.

**P_{1.5.2.2.12}**

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

P_{1.5.2.2.12}: IRS, User, Individual Tax Process, UI \[\rightarrow\] E-form 1040

JUSTIFICATION J1.5.2.2.12: Regarding the usability notion of throughput, the number of times a user is frustrated by the system will be 2 times or less per filing.
\[P_{1.5.2.2.13}\]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2

\[P_{1.5.2.2.13}: \text{IRS, User, Individual Tax Process, UI} \vdash \text{E-form 1040}\]

JUSTIFICATION \textit{J1.5.2.2.13}: Regarding the usability notion of throughput, the time for an experienced filer to complete Form 1040 in isolation of other forms and schedules will be under an hour.

<table>
<thead>
<tr>
<th>NFR</th>
<th>Type</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to show new form/screen</td>
<td>Performance</td>
<td>This should be 4 seconds or less.</td>
</tr>
</tbody>
</table>

\[P_{1.5.2.2.14}\]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.2.2.14

\[P_{1.5.2.2.14}: \text{IRS, User, Individual Tax Process, UI} \vdash \text{E-form 1040}\]

JUSTIFICATION \textit{J1.5.2.2.14}: Regarding the notion of performance, the time for the system to show a new web page will be 4 seconds or less.

\[P_{1.5.3}\]

STEP ID: Application of PROBLEM PROGRESSION to P1.5.3
\[ P_{1.5.3} \]: Individual Tax Process, UI E-form $\vdash$ 1040

JUSTIFICATION \( J1.5.3 \): This step simplifies the problem by removing domains not required to solve it. It allows concentration on the Functional requirements.

\[ P_{1.5.3.1} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.3

\[ P_{1.5.3.1} \]: IRS, User, Individual Tax Process $\vdash$ E-form 1040

JUSTIFICATION \( J1.5.3.1 \): The system needs to allow users the ability to send their completed tax return directly to the IRS using the IRS e-file standard format.

\[ P_{1.5.3.2} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.3

\[ P_{1.5.3.2} \]: User, Individual Tax Process $\vdash$ E-form 1040

JUSTIFICATION \( J1.5.3.1 \): The system needs to allow users to save their current session, with the intention of retrieving it in a subsequent session. The will be needed by some users but not necessarily all users.

\[ P_{1.5.3.3} \]

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.3
P\textsubscript{1.5.3.3}: User, Individual Tax Process $\vdash$ E-form 1040

JUSTIFICATION \textit{J1.5.3.3}: The system needs to allow users to retrieve a previous session for the tax year. The system must allow the users to continue work from the previous session.

\begin{itemize}
\item \textbf{P \textsubscript{1.5.3.4}}
\end{itemize}

STEP ID: Application of REQUIREMENT INTERPRETATION to P1.5.3

P\textsubscript{1.5.3.4}: IRS, User, Individual Tax Process $\vdash$ E-form 1040

JUSTIFICATION \textit{J1.5.3.4}: The system needs to allow users to send previously completed, and possibly paper based, associated tax forms to the IRS as part of the filing process. An alternative is to reference the form by an identifier and send the IRS the identifier.
Appendix B - UI Requirements

The requirements in this section were captured using UCD. This document considers a user to be someone who uses the solution to complete their individual tax return. Users that are out of scope are tax professionals and the IRS.

Domain

This section describes the knowledge that a user would need to use the solution:

- US Individual Taxation. Users need a high level understanding of how they are taxed, the processes and the parties involved. At a more granular level knowledge would include:
  - Different ways to submit tax return.
  - The types of tax form.
  - The terminology used on tax forms and by the IRS.
  - Who can assist completion of tax returns.
  - How, when and why money is exchanged with the IRS.

- Home computing, including an internet browser. To use the web based solution, a user would need to be comfortable with home computing.

Users

This section describes the characteristics of the user base for an electronic version of Form 1040.
Table 5 User Characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Form 1040 user characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>18 to death. Includes retired users</td>
</tr>
<tr>
<td>Sex</td>
<td>Male and Female</td>
</tr>
<tr>
<td>Culture</td>
<td>Primarily English speaking. Many Americans do have an alternative first language, most commonly Spanish.</td>
</tr>
<tr>
<td>Physical abilities and disabilities</td>
<td>Able bodied or may have limitations relating to hearing, sight. Some users will be colour blind.</td>
</tr>
<tr>
<td>Educational background</td>
<td>May have limited education, limited numeracy and/or limited literacy.</td>
</tr>
<tr>
<td>Computer/IT experience</td>
<td>May have little or no prior experience of computers/IT</td>
</tr>
<tr>
<td>Motivation</td>
<td>Many motivations, but not exclusively limited to:</td>
</tr>
<tr>
<td></td>
<td>• To minimise the tax paid.</td>
</tr>
<tr>
<td></td>
<td>• To ensure appropriate refunds are received.</td>
</tr>
<tr>
<td></td>
<td>• Legally obliged.</td>
</tr>
<tr>
<td></td>
<td>• Threat of penalties on late submission.</td>
</tr>
<tr>
<td>Attitude</td>
<td>Attitude may vary. Some comfortable using computers. Would expect that older users would be less comfortable using an electronic system over a paper based one.</td>
</tr>
</tbody>
</table>
The following table shows requirements that are directly derivable from user characteristics:

**Table 6 Requirements derived from User Characteristics**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>n/a</td>
</tr>
<tr>
<td>Sex</td>
<td>n/a</td>
</tr>
<tr>
<td>Culture</td>
<td>UI and documentation should be available in both English and Spanish</td>
</tr>
<tr>
<td>Physical abilities and disabilities</td>
<td>1. For the deaf, the UI should not depend on any audible feedback.</td>
</tr>
<tr>
<td></td>
<td>2. For the blind, the UI should be screen reader friendly.</td>
</tr>
<tr>
<td></td>
<td>3. The UI should provide visible and audible feedback.</td>
</tr>
<tr>
<td></td>
<td>4. For the partially sighted the UI would need a large font, and suitable contrast.</td>
</tr>
<tr>
<td></td>
<td>5. For the colour blind the system should use a colour scheme that allows these users to clearly see the contents.</td>
</tr>
<tr>
<td>Educational background</td>
<td>1. The system should be simple to use.</td>
</tr>
<tr>
<td></td>
<td>2. The system should be simple to learn.</td>
</tr>
<tr>
<td></td>
<td>3. The system should use simple language where possible.</td>
</tr>
</tbody>
</table>
User’s Work

The goal of users is to pay their yearly tax to the IRS.

The tasks involved in achieving this goal are:

1. To complete Form 1040 and associated forms.
2. Submit the tax forms to the IRS.
3. To make payments to the IRS, or to get refunds from the IRS.

The characteristics of completing Form 1040 task are:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance when task is repeated</td>
<td>May be variable:</td>
</tr>
<tr>
<td></td>
<td>• The government may change tax laws from one year to the next. It may introduce/revoke certain sections of the forms.</td>
</tr>
<tr>
<td></td>
<td>• Users whose circumstances do not significantly change from one year to another, may find form completion is similar to the year before.</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>Once per year.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| **Knowledge or user skills required** | An understanding of the components of Form 1040.  
An understanding of personal taxation in the US. |
| **Is time critical** | Yes:  
- Late submittal results in penalty charges.  
- Full information for the year is required to enter into the form, this information is by definition only available at the end of the year. |
| **Are their safety hazards** | No |
| **User is alone or has support of others** | Support is available from the IRS, Tax professionals, and potentially other tax payers – whether they are users or not. |
| **User is switching between tasks** | Potentially yes, completing the tax return may:  
- Be a long running task, and broken into several sittings.  
- Need access to information which is not to hand. |
The following table shows requirements that are directly derivable from task characteristics:

### Table 8 Requirements from Task - complete Form 1040

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance when task is repeated</td>
<td>There are many potential approaches to this.</td>
</tr>
<tr>
<td></td>
<td>Needs further investigation.</td>
</tr>
<tr>
<td>Frequency</td>
<td>n/a</td>
</tr>
<tr>
<td>Knowledge or user skills required</td>
<td>The UI needs to include a help system.</td>
</tr>
<tr>
<td>Is time critical</td>
<td>The UI needs to allow users to efficiently complete the form(s).</td>
</tr>
<tr>
<td>Are their safety hazards</td>
<td>n/a</td>
</tr>
<tr>
<td>User is alone or has support of others</td>
<td>The UI needs to allow a variety of parties to unambiguously identify different components within the Form.</td>
</tr>
<tr>
<td>User is switching between tasks</td>
<td>The system needs to allow users to store a current session, and allow them to return to the same session later.</td>
</tr>
</tbody>
</table>

An initial workflow analysis shows that in areas:
• The task does not need to be performed in a strict sequence. For example entering the label section first or last makes no impact on the outcome. It may also be that certain information is not currently available and needs to be deferred.

• The task is dependent on sub-tasks such as completing related schedules/forms.

Requirements derivable from workflow analysis:

Table 9 Requirements from initial workflow analysis

<table>
<thead>
<tr>
<th>Item</th>
<th>Requirements</th>
</tr>
</thead>
</table>
| Non-sequential | 1. The system does not need enforce a strict global workflow sequence.  
|              | 2. The system needs to provide feedback for tasks that are complete, incomplete and not started.                                              |
| Dependencies | 1. The system needs to also include electronic versions of schedules and forms.  
|              | 2. Some forms may be paper based and already exist, for these the system will need to either allow recording the sub form against Form 1040. This may be by reference number, a scanned copy or another. |

Usability Requirements

This section states the non-functional usability requirements for the solution. I have supplied an initial stab at measures, however I would expect to hone this in with the
customer and as yet undetermined supporting facts and figures. The choice of NFRs is based on Tyldesley 1998, as printed in OU UCD(2001).

**Table 10 Usability Non-functional Requirements**

<table>
<thead>
<tr>
<th>NFR</th>
<th>Type</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to complete Form 1040 in isolation of other forms and schedules.</td>
<td>Throughput</td>
<td>If a user has all necessary referenced documentation to hand, completion time under an hour.</td>
</tr>
<tr>
<td>Time to show new form/screen</td>
<td>Performance</td>
<td>This should be 4 seconds or less.</td>
</tr>
<tr>
<td>Number of times user has to refer to a third party for advice.</td>
<td>Learnability</td>
<td>This will be dependent on the complexities of the particular users. On average one enquiry to the either the IRS or tax professional per 10 simple users.</td>
</tr>
<tr>
<td>Time to commence entering data into the solution.</td>
<td>Availability</td>
<td>Meaning not needing to install the application on a local computer. The system will be available for usage 24/7 for 178 concurrent days a year.</td>
</tr>
<tr>
<td>Number of times user referenced documentation not included with the solution.</td>
<td>Learnability</td>
<td>Should be less than 1 per 15 users.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Number of times the UI misled the user.</td>
<td>Throughput</td>
<td>2 times per usage.</td>
</tr>
<tr>
<td>Percentage of users who would recommend the solution to a friend.</td>
<td>Attitude</td>
<td>70%</td>
</tr>
<tr>
<td>Number of users preferring this solution compared to previously used solutions.</td>
<td>Attitude</td>
<td>40%</td>
</tr>
<tr>
<td>Number of times user is frustrated by the system.</td>
<td>Throughput</td>
<td>2 times per usage</td>
</tr>
</tbody>
</table>
Appendix C – Level Test Plan

This appendix holds a acceptance level test plan conformant to IEEE 829-2008 standards.

Introduction

Document ID: TPL-Acceptance

Scope: this test plan is contains acceptance level testing for Form 1040.

References:

Testing Level: Acceptance

Overall Test Conditions: The objective of these tests is to prove that the solution meets the system requirements.

Details

Test items and their identifiers: TBD

Test Traceability Matrix: TBD

Features to be tested:

Features not to be tested: TBD

Approach: Black-box testing

Item pass/fail criteria: Any item that demonstrates an incorrect calculation is a failure.

Other items will be determined on a case by case basis.
Suspension criteria and resumption requirements: TBD

Test deliverables:

• Level Test Design(s)
• Level Test Cases
• Level Test Procedure(s)
• Level Test Report
• Anomaly Report

Test management

TBD
Appendix D - Level Test Design

This appendix holds a acceptance level test design conformant to IEEE 829-2008 standards.

Introduction

Document ID: LTD-Acceptance-Income

Scope: System level tests for the income section of the form 1040 solution.

References: Beizer(1995)

Details of the Level Test Design

Features to be tested: Income Section

Approach Refinements: Data Flow Testing – a black box technique. At risk of oversimplifying the problem under test, the solution is calculator and therefore the correct flow and use of data is critical to correct calculation.

Test identification:

Beizer Model

The following model is copied from Beizer (1995) and updated for the 2007 version of the same form. It follows Beizer’s textual model to represent data flow testing, what he calls “linked list notation”.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Links to Line number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:</td>
<td>22:</td>
<td>wages</td>
</tr>
<tr>
<td>8a:</td>
<td>22:</td>
<td>taxable_interest</td>
</tr>
<tr>
<td>8b</td>
<td></td>
<td>tax_exempt_interest</td>
</tr>
<tr>
<td>9a</td>
<td>22:</td>
<td>oridinary_dividends</td>
</tr>
<tr>
<td>9b</td>
<td></td>
<td>qualified_dividends</td>
</tr>
<tr>
<td>10</td>
<td>22:</td>
<td>taxable_refunds</td>
</tr>
<tr>
<td>11</td>
<td>22:</td>
<td>alimony_received</td>
</tr>
<tr>
<td>12</td>
<td>22:</td>
<td>business_income_or_loss</td>
</tr>
<tr>
<td>13</td>
<td>22:</td>
<td>capital_gain_or_loss</td>
</tr>
<tr>
<td>14</td>
<td>22:</td>
<td>other_gains</td>
</tr>
<tr>
<td>15a</td>
<td>15b:</td>
<td>ira_distributions</td>
</tr>
<tr>
<td>15b</td>
<td>22:</td>
<td>ira_distributions_taxable_amount</td>
</tr>
<tr>
<td>16a</td>
<td>16b:</td>
<td>pensions_annuities</td>
</tr>
<tr>
<td>16b</td>
<td>22:</td>
<td>pensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_annuities_taxable_amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>17</td>
<td>22:</td>
<td>rental_royalties_etc</td>
</tr>
<tr>
<td>18</td>
<td>22:</td>
<td>farm_income_or_loss</td>
</tr>
<tr>
<td>19</td>
<td>22:</td>
<td>unemployment_compensation</td>
</tr>
<tr>
<td>20a</td>
<td>20b:</td>
<td>social_security_benefits</td>
</tr>
<tr>
<td>20b</td>
<td>22:</td>
<td>social_security_benefits_taxable_amount</td>
</tr>
<tr>
<td>21</td>
<td>22:</td>
<td>other_income</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>total_income := wages + taxable_interest + …. + other_income</td>
</tr>
</tbody>
</table>

Tests Required:

1 test for each addition to the total_income (line 22), with all other input values set to 0.

1 test to demonstrate that the total_income is adding all values together. Each line added to total_income is set to 1.

1 test for each non-included field of the total_income calculation, all other input values set to 0.
Feature pass/fail criteria: All tests must pass. In a calculation it is the overall outcome that is the most important factor, any demonstration of not producing the correct output value represents a serious system problem.

Test deliverables:

- Level Test Cases
- Level Test Procedure
- Level Test Report
- Anomaly Report
Appendix E - Test Cases

This appendix holds a acceptance level test cases conformant to IEEE 829-2008 standards.

Introduction

Document ID: TCA-Income

Scope: This document contains all acceptance level test cases that pertain to the income section of Form 1040.

References: Form 1040 – POE specification.

Test Case ID: TC-Income1

Objective: to show that wages are added to the total_income field.

Inputs:

All fields except wages set to 0.

Wages set to 1.

Outcome:

Total_income field shows a value of 1.
Test Case ID: TC-Income

Objective: to show that each of the remaining 14 fields can be individually set.

Repeat this test in turn for each of the 14 fields.

The 14 remaining fields are:

1. taxable_interest
2. ordinary_dividends
3. taxable_refunds
4. alimony_received
5. business_income_or_loss
6. capital_gain_or_loss
7. other_gains
8. ira_distributions_taxable_amount
9. pensions_annuities_taxable_amount
10. rental_royalties_etc
11. farm_income_or_loss
12. unemployment_compensation
13. social_security_benefits_taxable_amount
14. other_income

Input:

All fields except the field under test set to 0.

Set the field under test to 1.

Output:
For each test the total_income field shows a value of 1.

Test Case ID: TC-IncomeAllIncluded

Objective: to show that all appropriate fields are added to the total_income field.

Input:

Set all of the following 15 fields to 1. Field names:

1. wages
2. taxable_interest
3. oridinary_dividends
4. taxable_refunds
5. alimony_received
6. business_income_or_loss
7. capital_gain_or_loss
8. other_gains
9. ira_distributions_taxable_amount
10. pensions_annuities_taxable_amount
11. rental_royalties_etc
12. farm_income_or_loss
13. unemployment_compensation
14. social_security_benefits_taxable_amount
15. other_income

Outcome:

Total_income field shows 15.
Test Case ID: TC-IncomeCorrectExclusion

Objective: to prove that any field that is not part of the total_income calculation, is not included in the calculation.

Input:

Set each of the following excluded fields to 1. Fields: tax_exempt_interest, qualified_dividends, ira_distributions, pensions_annuities, social_security_benefits

The remaining fields are set to 0.

Outcome:

Total_income field shows 0.
Appendix F - Test Procedure

This appendix holds the test procedure required by previously stated test cases, conformant to IEEE 829-2008 standards.

Introduction

Identifier: TP-Income1

Scope: supports all income acceptance level test cases.

References: income acceptance level test cases.

Relationships to other procedures: none

Details

Inputs and outputs: as determined by test cases.

Steps to execute:

1. Access the income section of the solution.
2. Set all fields as defined in the test case.
3. Observe the correct output, as defined in the test case.

General

n/a
Appendix G – Non-Functional Requirement Tests

NFR Test 1

Requirement under test: “The percentage of users who would recommend the solution to a friend will be at least 70%”.

Test pass/fail criteria: as above.

This section presents an outline for testing above requirement.

The Evaluation Outline

Assume a sample of 10 users will give statistically significant results.

Ask each user to perform their tax returns on the prototype system.

On completion ask the users to fill a questionnaire; one of the questions will be a qualitative Likert scale showing the likelihood of the user recommending the system to a friend.

It would be sensible to also ask users to say what aspects they liked most and least about the system. This would be useful feedback to direct further work.

NFR Test 2

Requirement under test: “Regarding the notion of performance, the time for the system to show a new web page will be 4 seconds or less.”

Test pass/fail criteria: as above.
There are numerous ways that this could be tested; I present an outline for just one of them here.

Test Outline:

Using a load testing tool:

- Take a recording of a user accessing each web page.
- Run the load testing tool against a server that’s equivalent to the final production server.
- Step the number of users up to 500 concurrent users, each playing the recording provided in the first step.
- The pass criteria is that under full load, all pages are returned in 4 seconds or less.
Appendix H – Constraint Requirement Tests

Requirement under test: “The UI must be available in both English and Spanish.”

**Objective:** to show that every word displayed is consistently in English for one case, and Spanish for the other. This includes associated online help.

**Test pass/fail criteria:** that all text is in the correct language, exceptions allowable are proper names, or financial terminology where translation results in less clarity for a native speaker. Each language can pass or fail.

**Testing the User Interface**

For each language, a native speaker given screen shots (prototypes would be acceptable) of every screen, and asked to identify all words that are not in the correct language.

**Testing the Online Help**

The assumption here is that the help will not be available at the same time as the UI prototypes, therefore the test will be performed at another time. A native speaker will review all of the online help, and again asked to identify all words not in the correct language.
Appendix I - Abbreviations

ABJ - Assurance-Based justifications

NFR – Non-Functional Requirement

POSE – Problem Oriented Software Engineering

TBJ – Test-Based Justification

UCD – User Centred Design