

IMS Systems
Quality Assurance Procedure
Project Documentation

Project Documentation Procedure (QA-TP01)

1. Introduction

This procedure covers the general aspects of the overall Quality Procedure used for software development and maintenance by IMS Systems. Its main purpose however, is to identify the documentation used in the maintenance of quality, thus providing a method of control over the continuing development of the companies products. Documentation requirements detailed in this and other procedures are necessary to provide an audit trail.

Reference to software development in the Quality Assurance Procedures includes new projects, upgrading of exiting systems and maintenance of systems.

1.1 Scope

The scope of this procedure covers the production of the following documents, many of which will be covered in detail in the appropriate procedure as referenced.

- 1.1.1 Procedure layout, ratification and numbering
- 1.1.2 Document Distribution
- 1.1.3 Forms Control & numbering
- 1.1.4 Responsibility Identification
- 1.1.5 Project Plan
- 1.1.6 Project Diagrams: Modular Tree, Relationship Model, Class Diagram
- 1.1.7 House Style
- 1.1.8 SourceSafe history printouts (see Software Testing)
- 1.1.9 Software Failure Card (see Software Testing)

2. Procedure Layout, Ratification and Numbering

This procedure demonstrate the layout used in the formal Quality procedures. Standard paragraph number based on the legal method is used. Paragraph headings are in a bold face. The typeface used is Times New Roman at 12 point. The title of each procedure is shown on the title page, the top of the first page of the procedure text and in the footer of each page. The footer also contains the page number with the title page being page 1, there after running sequentially on single sided printing. The header of each page contains the company's name and the heading 'Quality Assurance Procedure QA-TPXX' where 'TP' stands for Project Code, and XX is the procedure number.

Procedures should be ratified by two senior members of the company, one being a director. The ratification signatures and date will be at the bottom of the last page of the procedure. Any amendments to procedures will require their re-issue and ratification, hence the version will be

in chronological order of the ratification date, the most recent being the current version.

2. Document Distribution

All procedures should, after ratification, be copied and distributed to all members of the project team and senior personnel. A master set must be kept with the project documentation. In effect the documents referred to in this procedure form the project documentation and should be kept chronologically in an ordered manner based on software units, modules and system.

3. Forms Control and Number

Any forms created should be referenced in an addendum to this procedure by use of a 'Forms Design Number Control' register (copy attached). Full details of the use of any form must be made in the appropriate procedure and formally accepted by a senior manager before they are taken into practice. The creation on any form will require the amended procedure to go through the ratification process.

The following forms have already been referenced in the Software Testing procedure, copies will be included as they become available. (This is an interim measure during the development of the Quality procedures).

- 3.1 Software Development Control Sheet (see Software Testing)
- 3.2 Software Test Log (see Software Testing)
- 3.3 User Test Control Sheet (see Software Testing)
- 3.4 User Software Failure Sheet (see Software Testing)

4. Responsibility Identification

Before the start of any project or update of existing systems, individual responsibility must be identified.

Requirements, tasks and deliverables must be registered regardless of how small it may be.

Each person given a task will be responsible for the completion of that task.

The responsibility for testing a module will be assigned to a member of the project team not responsible for the coding of that module.

All above responsibilities will be logged with the General Manager and shown on appropriate documentation.

5. Project Plan

All projects will require a project plan. The plan should cover all aspects of software development which influence quality; there should be no activity or action which cannot be related to the plan. It may be a single document, or may comprise of several documents each covering a particular aspect of the plan e.g. programming standards, configuration management.

5.1 The project plan should include:

- 5.1.1 purpose and objectives of the project;
- 5.1.2 definition and description of project stages (or phases), e.g. description of project design;
- 5.1.3 identification of the products of each stage including specification, documentation, test schedules, program code;
- 5.1.4 overall quality requirements and objectives for the project;
- 5.1.5 each stage or phase should mark a significant and clearly identifiable step in the project;
- 5.1.6 deliverables to the customer or user should be clearly identified.

In addition to the above the detail of the plan should:

- 5.1.7 identify the project activities, their function and objectives;
- 5.1.8 show personnel job titles, tasks and responsibilities;
- 5.1.9 identify the interface between activities and between jobs;
- 5.1.10 identify quality assurance tasks and responsibilities.

5.2 Project Plan Format

The format of the project plan should include a task over time chart e.g. Gantt chart, it should also identify any task dependencies. The project plan will be updated at agreed regular intervals and distributed to a members of the project team. In a supporting written document the plan should cover:

- definitions of the project;
- organisation and structure;
- standards and conventions to be adhered to;
- techniques and methodologies used;
- review and audit meetings;
- configuration management tools and methods;
- change control procedures;
- documentation requirements.

6. Project Diagrams

All software development must be supported by appropriate project diagrams. It is not suggested here that a full set of diagrams down to 'Procedure Specification' i.e. decision trees or flowcharts, are required, however, there may be occasions when this is felt to be necessary. Diagrammatic conventions are to be agreed at the start of a project and supported by a standards paper. Such standards papers will form part of the QA documentation.

A minimum of the following diagrams must be produced:

- Modular Tree
- Data Flow Diagrams
- Entity Relationship Model
- Entity Life Histories
- Class Diagram (OOD only).

These diagrams (they may consist of sub-diagrams) will provide the basis for building suitable test plans, the start of a data dictionary and a quality check on the system requirements.

6.1 Modular Tree

The modular tree should show all modules of the system, their place in the hierarchy, iteration and selection.

6.2 Data Flow Diagrams

This shows the movement of data through the system, identifying processes and data stores. External Entities are shown at Level 0 and Level 1.

6.3 Entity Relationship Model

This shows the relationship between each module and can be supported by a logical data structure diagram and data access diagram if it is felt necessary by senior management.

6.4 Entity Life Histories

Providing an operational life of each entity and the events that take place during its existence.

6.4 Class Diagram

This will show each class defined, the class name, access identifiers, member functions,

data members and association.

7. House Style

All coding will conform to an agreed House Style as set out in the document 'Coding Standards'.

The IMS Systems House Style has been developed, thus making it easier for Analysts, Designers and Programmers to understand and read the programs developed. Use of a House Style makes the debugging and maintenance of programs easier. Failure to maintain and use the House Style will result in Disciplinary Procedures being taken.

7.1 Style Standards

7.1.1 Comments

The 'Program Information' comment section at the top of the template must be complete and include the programmers name. All variable declarations should be accompanied by a descriptive comment. Function definitions should have the preceding 'Function' comment box completed to provide a description of the function. The program body should be annotated by comments where this makes the program clearer. Comments should be clear and to the point.

7.1.2 Declaration of Functions

When declaring a function prototype the function name should use capital letters for the first letter. If the name consists of two or more words, each word can start with a capital letter,

eg `void DriveMarco(void);`

7.1.3 Declaration of variables

All variables should again start with a capital letter, if the variable name is made up of more than one word, each word can start with a capital letter,

eg `int JoyStick;`

Hungarian notation will not be accepted.

7.1.4 Preprocessor defines

Preprocessor #define statements must be in all upper case letters,

```
eg    #define MOTORFORWARDS 10
or    #define MOTOR_FORWARDS 10
```

7.1.5 Indentation

The program must be properly indented, with each structure or block being indented three spaces inside the last.

After the variables are declared the main structure starts 3 spaces from the left, the following structure (starting with the 'while' statement) will be indented after its first statement, the brace '{' for opening it on the same line as the statement. The closing brace '}' will be on a line of its own and line up with the indentation of the first line, all lines between are indented by 3 spaces. This applies to all structures and a more complex example is given in the KeyPress function using 'if/else' and 'if' structures.

7.1.6 Compound statements

In if, if else, switch / case, for, while and do while structures the body of the structure must be in all cases a compound statement, i.e. enclosed in {}.

7.1.7 Blank lines

A blank line should follow the declaration of variables and immediately after each structure. Additional blank lines can be used to assist in the readability of the code.

7.1.8 Operators

Binary operators should be separated from their operands by one space,

```
eg    Key = getch();
       Average = (Num1 + Num2 + Num3) / 3.0;
```

7.1.9 Global variables

Global variables should be avoided if possible and be kept to an absolute minimum..

7.2 House Style Example

The program on the following pages uses the **TEMPLATE.C** skeleton for its bases, this must be used for all your programs. The program is written to demonstrate the House Style required, therefore comments are used for this purpose and not for code explanation.

```
//===== Program Information =====

//NAME OF SOURCE FILE:  H_STYLE.C

//AUTHORS NAME:          A.N.OTHER

//PROGRAM DESCRIPTION:  A short program that calculates the VAT on a
//                      given value to demonstrate House Style

//DATE:                  20/10/96

//LIBRARIES
#include <stdio.h>
#include <rack.h>

//DEFINITIONS
#define VAT              17.5      // represents VAT rate as percentage

//GLOBAL VARIABLES

//FUNCTION PROTOTYPES
char KeyPress(void);           // Initial capitalised of each word

//===== MAIN BLOCK =====

void main(void){
    float NetValue;           // variable for net value of Vat and
    float VatValue;          // VAT amount of sum
    char Key = {'a'};        // character returning option from menu

    clrscr();                // All subsequent lines and structures indented by
                            // 3 spaces - see note 3 for clarification.
    while(Key !='X'){

        printf("\nEnter the net value on which VAT is required? £");
        scanf("%f",&NetValue);

        VatValue = NetValue * VAT / 100.0;

        printf("\n\nNet Value is £%.2f", NetValue);
        printf("\nVAT is £%.2f",VatValue);
        printf("\nGross is £%.2f",NetValue+VatValue);

        Key = KeyPress();
    }
}

/* *****FUNCTION*****
 * Function Name:  char KeyPress(void)
 *
 * Operation:     Gets a character from keyboard to continue or exit
 *
 * Parameters Passed:  None
 *
 * Return Values:  Choice - if = X then exit program
 ***** */
```

```
char KeyPress (void){
    char Choice;

    printf("\n\nPress any key to continue");
    getch();

    while(Choice != 'X' && Choice != 'C'){
        printf("Press X to exit or C to continue");
        Choice = getch();
        Choice = toupper(Choice);

        if(Choice == 'X'){
            printf("\nReturning to System - Please wait\n");
            delay (1000);
        }else{
            if(Choice != 'X' && Choice != 'C'){
                printf("\nPlease select 'X' or 'C' - try again!\n");
            }
        }
    }
    clrscr();
    return(Choice);
}
```

8. SourceSafe

Current configuration management of source code is maintained by an application called 'SourceSafe'. This will continue to be used, however, a hard copy of the history report will be produced for inclusion in the project documentation to provide an audit trail and support testing.

9. Software Failure Cards (SFC)

These documents are currently held in text (ASCII) files. They should be printed out and held with other project documentation and reference the SourceSafe history to which that version applies.

This procedure is ratified and taken into practice on this day: _____ Date: _____

Signed: _____

Name: _____

on behalf of IMS Systems.