QUALITY ASSURANCE IN BUSINESS SIMULATION DESIGN

Jeremy J. S. B. Hall
Hall Marketing
jeremyhall@simulation.co.uk

ABSTRACT

This paper explores the elements of business simulations that impact software (model) quality rather than learning quality. The exploration draws on the computer software knowledge base and extends this to take into account the special characteristics of business simulation software. Business simulation design is a creative art where the simulation models are complex and where the users are extremely wide ranging with limited knowledge of the simulation software and are commonly very emotionally involved - issues that necessitate a high level of software quality. Business simulation model complexity is explored in terms of model size, arithmetic calculations, cyclomatics, structure and dynamics. Error types are those normally associated with software (syntax, run-time and logical) and require testing using of black-box (functional) testing, white-box (structural) testing, code inspection and, in additional, for business simulations structural and dynamic testing. But, as quality cannot be tested into the simulation, Total Quality Management is vital and explored in terms of methodology, software structure, modelling language, defensive programming, refactoring, documentation and verification support

Keywords: quality assurance, model verification, complexity, error types, testing, quality management.

INTRODUCTION

The heart of a business simulation is a model that attempts to replicate the real world and consists of arithmetic and logical statements. Besides the simulation model a business simulator has other software components (Hall, 2011) whose purpose are to manage software use, decision entry, reporting and, possibly, online help. However, this paper concentrates on how one can assure and verify the quality of the simulation model (software) rather than the other software components or the ability for the simulation to deliver quality learning. That is to say the paper explores *verifying* that the software-based model performs as intended rather than *validating* that the software fulfils its intended purpose (Law & Kelton, 1991). A particular issue is the conflict in all software design between *engineering design* and *creative design* (Löwgren, 1995) and, arguably, for business simulation this conflict is worse because of creative needs and aggravated by the use to provide business learning to a wide mix of users who have an emotional involvement. The creative design process hampers the ability to design quality software and the emotional engagement of participants amplifies the impact of poor quality.

Designing a business simulation is a creative art (Bellman et al, 1957; Goosen, 1981; Thavikulwal, 2004; Bots & Daalen, 2007) where the simulation (model, decisions and results) are built in an agile, iterative-incremental process (Hall, 2005). The way the model and associated data (variables, reports and help screens) grow over time during the development of a complex business simulation (the Training Challenge simulation) is shown in Figure 1.

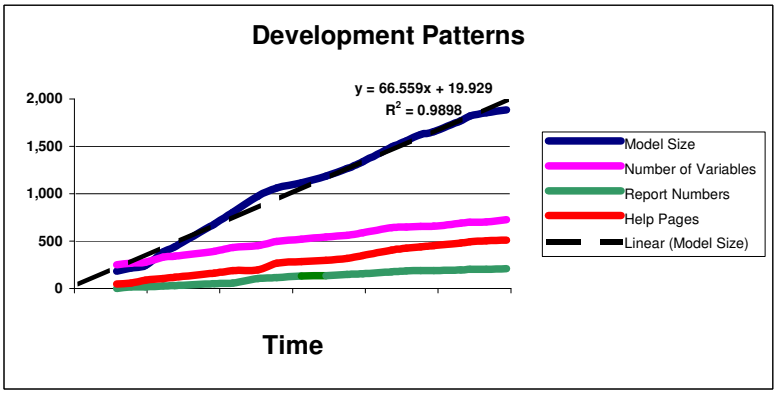

Figure 1: Simulation size growth during development

The model size, variable and report number patterns over time show that these change in concert *throughout* the design and this suggests an incremental progressive design process. It seems reasonable to suggest that if the process was not incremental there would be a long period at the start of the design where the parameter, decision and report needs were defined before the model was programmed. Later, as the models linking decisions to results were developed, there would be minor increases to the number of variables, reports and help pages. Where software is designed is an *agile, lightweight* way the requirements are emergent (discovered during the project) and a particular weakness of this approach is *"poor overall quality"* (Khan et al, 2011).) Mohammad et al (2013) cites *"poor documentation"* as an another weakness of agile design and that this has implications in terms of software testing, maintenance and communicating with users.

There are several usability issues - decision scope, the range of users and user engagement. Participants have and need virtually unlimited authority with their decision-making and this means that the decisions entered into the simulation can range from the reasonable to the unreasonable. As a consequence, the designer must ensure that even the unreasonable decisions do not *"break"* the model. The simulation will be used by a wide range of participants and tutors who will have differing levels of business knowledge, computer literacy and, probably, no or minimal knowledge of the simulation software. This range of users will stress the model's quality as they may make mistakes when entering decisions. The way business simulations engage their participants is recognised but there is a downside, if something goes wrong this *breaks engagement* (Aldrich, 2009) leading to disaffection. Although this is true for other software (attempting to enter data into a badly designed web form comes to mind) with simulations a problem does not just impact an individual in private, rather it impacts several, perhaps, many learners in a very public way. Even if the problem is resolved, trust in the simulation and it's ability to deliver learning is likely to be destroyed both for the participants and for the tutor using the simulation.

COMPLEXITY

Business Simulation models are complex software and the are several aspects of complexity (model size, arithmetic complexity, cyclomatic complexity, structural and dynamic complexity) that impact quality.

MODEL SIZE

Hall (2007) investigated model size, parameter and report numbers for several business simulations. Four of the simulations Hall explored had a duration of about a day and their model sizes ranged between 964 and 2086 statements, used between 476 and 588 variables and produced between 112 and 209 different reports. A wider analysis of twenty simulations with durations ranging from two hours to two and a half days found model sizes ranged in size from 271 statements to 2127 statements. These simulations were created using Visual Basic and for models created in Excel the model size would be substantially larger because in terms of function points (QSM - Function Point Languages Table Version 5.0 2013) Excel typically requires five times the number of lines of code. This means that, if the models were developed using Excel the model sizes could range from around 1000 to about 10000 lines of code (and this does not include the lines of code for data storage, reporting, decision-entry etc.). The model size metrics for Excel based business models are relevant as their widespread use means that their error rates have been researched extensively. Freeman (1996) suggest that spreadsheet models with more than 150 rows (logic lines) has at least one *significant* error and others have raised major concerns about errors (Cook, 2006; Howard, 2005; Panko & Halverson, 1996; Panko, 2000; Rajalingham et al, 2000)). Although this error rate is partly due to the nature of spreadsheets it is also influenced by model size, arithmetic, cyclomatic, structural and dynamic complexity.

ARITHMETIC COMPLEXITY

Calculations in business simulations range from basic accounting and operational calculations to complex non-linear calculations and, on occasion, stochastic calculations. A business simulation's core models can be separated into the *"white box"* models that replicate the basic accounting, work flow, supply chain elements, etc. and the *"black box"* models that model consumer behaviours (Kotler, 1991), staff effectiveness and efficiency, etc.. The complexity of these models have been described in numerous ABSEL papers (Gold & Pray, 1990; Teach & Schwartz, 2000; Murff et al, 2006; Goosen, 2007 and others). The simplicity of the white box models means that they are unlikely to be mistyped. Whereas, the complex, non-linear nature of the black box models is likely to lead to errors during programming and testing.

CYCLOMATIC COMPLEXITY

McCabe (1976) identifies software that *"will be difficult to test"* with a cyclomatic complexity metric that measures complexity in terms of the number of possible paths through the software. Hansen (1978) clarifies this in terms of logical

complexity (IF THEN statements, CASE statements and WHILE/UNTIL statements.) In response to decisions a business simulation will use different paths to calculate impact. A review of 20 business simulations with model sizes ranging from 271 statements (with 36 paths) to 2127 statements (with 434 paths) suggested a linear relationship between the number of statements and the number of paths (Figure 2: Cyclomatic Complexity relative to Model Size) and this linear relationship and correlation duplicates that found by Shepperd (1988). Cyclomatic complexity is relevant as complex paths are difficult to visualize and, ideally, every path needs to be tested (McConnell, 2004). The need to test paths impacts testing time and where the time available for testing is limited this precludes testing all paths increasing the chance of software defects.

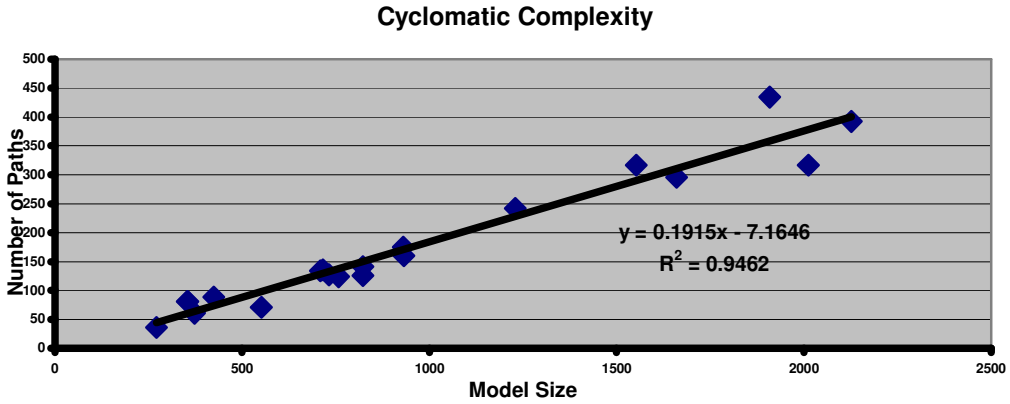

Figure 2: Cyclomatic Complexity relative to Model Size

The empirical study of the impact of cyclomatic complexity by Gill & Kemerer (1990) investigated a range of projects with cyclomatic complexity ranging from about 0.107 to about 0.197 and where the maintenance productivity of low complexity projects was four to eight times higher than complex projects. As illustrated in Figure 2, the cyclomatic complexity of 0.192 is at the top of Gill & Kemerer's study meaning that maintenance (and testing) effort will be significant.

STRUCTURAL COMPLEXITY

A business simulation models several processes that interact and overlap. Typically a business simulation will model marketing (how the marketing mix drives sales), operations (the ability to provide for sales based on capacity and resources), a sales model (linking marketing and operations), an Income Statement model (calculating revenues and costs), a Balance Sheet model (calculating asset, equity and liability changes) and a Cash Flow Model (linking the Income Statement and Balance Sheet). The simulation must model these processes in a appropriate order. For example, although the marketing model outcomes can be determined before or after the operations model outcome, the sales model must be placed after both the marketing and operations model. But besides the order of processing, models overlap. For instance, for the financial models (Income Statement, Balance Sheet and Cash Flow) some of the calculations are embedded in the earlier models. The Income Statement model needs to be split with revenue and costs calculated and used to determine bank funding needs before Financial Expenses can be determined. A further structural complexity is that often the simulation involves several markets and/or products that are processed in parallel. This means that the structure is complex and design and testing are difficult.

DYNAMIC COMPLEXITY

Typically a business simulation involves making decisions and receiving results on a period-by-period basis. This is a dynamic feedback process where the current period's outcomes (results) depend on the current period's decisions and the prior periods decisions and results (Hall & Cox, 1993). This dynamic process is at two levels - the dynamic behaviour of the business (the simulation model) (Gold, 2003) and the dynamic behaviour of the decision-making process. Both the dynamics of "real-world" business (Forrester, 1958) and the simulated world (as explored by the Beer Game (Goodwin & Franklin Sr., 1994)) can lead to instability leading to the "bullwhip" effect (Lee et al, 1997).

The dynamics complexity of a business simulation can range from simple where few interactions between variables and over time to the situation to where there are *"strong feedback loops and may be non-linear in nature. [And] There may be delays and inertia in the production, sales and distribution of products"* (Gold, 2003). This is exacerbated by *noise* (random events) and sudden changes breaks in economic patterns (Hall & Cox, 1993). At the extreme, dynamic complexity can lead to the situation where the learners lose control of their business - resulting in the inability to identify and manage cause and effect and becoming disaffected.

TYPES OF ERROR

The first step in assuring quality is an understanding of error types so as to design them out. Software errors fall into several classes (IEEE, 1990):

1. SYNTAX ERRORS
2. RUN-TIME EXCEPTIONS
3. LOGIC ERRORS

SYNTAX ERRORS

These are the errors associated with grammatical or structural rules of the language used to program the model (IEEE, 1990). They are generally not a concern as the compiler, interpreter or development platform will usually flag them as the model is created or when the software is compiled.

RUN-TIME EXCEPTIONS

These are the errors that become apparent when the simulator is used and halt the program (IEEE, 1990). Run-time exceptions include divide by zero and (depending on the modelling language) overflow, assignment to the wrong type of variable, infinite loops, buffer overflows, etc.. If run-time errors are not anticipated, detected and eliminated during design they are disastrous for when the simulation is used the simulator will *crash* and learning session will be disrupted or terminated. Even if recovery is possible, the learners are likely to become disaffected and lose trust in the simulation. Run-time errors are likely to be caused by extreme decisions or unexpected combinations of decisions. For example, learners decide to withdraw from a market resulting in zero sales for the market and its gross profit percentage calculation resulting in a divide by zero exception. Another example is where a simulator searches for contracts that match a set of criteria. If the criteria are too tight the search would continue for ever (infinite loop). A final example involves developing new products and adding them to the product range. As there will be a pre-defined maximum number of data fields for the products, buffer overflow will occur if the number of products exceed this.

LOGIC ERRORS

These are the errors associated with incorrect and missing arithmetic calculations, conditional logic errors and processing sequence errors. Errors that produce the wrong results but do not terminate the program and as they lead to the model behaving wrongly they are more insidious than Run-Time Exceptions. Biggs & Halpin (2004) give an example of an arithmetic calculation error was a simulation had an upwardly sloping demand curve resulting in sales increasing as price increased. In the same paper Biggs & Halpin describe a logic (path) error where there were three production constraints. Here testing covered paired comparisons but did not test for the situation where all three constraints were breached and this allowed excessive production when all three constraints were breached. An example of missing logic in the same paper was because there was no check on the number of sales people who could be fire, it was possible to fire more sales people than were employed. Consequentially, the simulation allowed negative sales staff and negative costs.

QUALITY ASSURANCE TESTING

Understanding sources of errors (error types) helps reduce their occurrence but ultimately the software needs to be tested and there are several types of testing:

BLACK BOX TESTING
WHITE BOX TESTING
CODE INSPECTION
STRUCTURAL SOUNDNESS TESTING
DYNAMIC STABILITY TESTING

Code Inspection, Black Box and White Box Testing are standard for all software. But the special nature of business simulations suggests that there are two further needs - Structural Soundness Testing and Dynamic Stability Testing.

BLACK BOX TESTING

Black Box (functional) testing is *"testing that ignores the internal mechanisms of a system or component and focuses solely on the outputs generated in response to selected inputs"* (Gao et al, 2003). For business simulations it involves checking that the decisions produce the correct results. A particular problem with black box testing is the difficulty determining the cause or causes of an error (which logic or arithmetic statements are the cause). For instance, if the Balance Sheet does not balance, this can be due to cash flow or costing errors or both or incorrect opening Balance Sheet data.

WHITE BOX TESTING

White Box (structural) testing *"takes into account the internal mechanism of a system or component"* (IEEE, 1990). In other words it involves tracing paths and how variables change as one *steps through* the model. The problem with white-box testing is that with some modelling languages it may be difficult or not possible to *step through* the model and explore how variables change.

CODE INSPECTION

Code Inspection involves visually inspecting the model's source code statement-by-statement (Myers et al, 2004). As discussed later, the modelling language has a major impact on the ease, speed and quality of code inspection.

STRUCTURAL SOUNDNESS TESTING

Structural soundness for business simulations means ensuring that data is correctly transferred between periods, parameters are initialised correctly, calculations are done, in the right sequence and are not done in multiple (and different ways). Although this should be covered by black and white box testing and code inspection, the structural complexity of the model and the problems associated with this means that it is advisable to check structural soundness separately.

DYNAMIC STABILITY TESTING

Dynamic testing involves exploring the simulation's dynamic stability. Ensuring dynamic stability is difficult to test because, often, it depends on participant perceptions and whether they "overreact". Dynamic testing takes place at two points - during simulation design and during piloting (alpha and beta testing). During design, stability testing involves testing to see how *extreme* decisions impact results and exploring the delays between decisions and outcomes. For instance, a major issue for a distribution company selling to other companies is to have sufficient inventory to service demand. If there are inventory shortages, not only does the company loose sales but, over time, will gain a reputation for poor customer service - a reputation that will reduce future sales. But as inventory value impacts profitability there is a desire to cut inventories. During, the alpha testing of a distribution industry simulation, it was found that it was too easy for participants to get a bad customer service reputation that was impossible to recover from. An example of the impact of delayed outcomes is where price cuts take time to be apprehended by customers and, to stimulate demand, participants may cut prices further.

TOTAL QUALITY MANAGEMENT

Testing cannot be exhaustive (Myers et al, 2004) and for business simulations this is particularly true because the model size and complexities mean that the time needed to test comprehensively is unacceptably long and costly. Ensuring quality must be an integrated, systematic strategy (Evans & Dean, 2002) - Total Quality Management (TQM). For business simulations TQM involves building quality into the design process and entails:

DESIGN METHODOLOGY
MODELLING LANGUAGE
SOFTWARE STRUCTURE
REFACTORING THE MODELS
DEFENSIVE PROGRAMMING
DOCUMENTATION
VERIFICATION SUPPORT

There are economic considerations of business simulation quality management. Good quality management during design serves to speed design and reduce the need for testing and through this help reduce development costs. Further, good design quality management, by providing good user documentation and reports that explain calculations and the impact of decisions, supports the business simulation's use in the classroom and so its cost is further mitigated by usage benefits.

DESIGN METHODOLOGY

Business Simulation design methodology can draw on instructional design methodologies (such as ADDIE (Molenda, 2003) or the Dick & Carey Model (Dick et al, 2008)) and computer software design methodologies (heavyweight (E.G. Waterfall) or lightweight (E.G. Agile)). Hall (2005) described a special business simulation design methodology (the Rock Pool method) that combined a heavyweight, rigorous process with lightweight agility to attempt to ensure that the new simulation is delivered to time and to cost without constraining creativity. In this methodology, the software design stage consists of two linked *rock pools* - design and development. The design stage involved creating models, deciding decisions and results, developing preliminary documentation and creating validation and quality assurance support. The development stage comprised testing and calibrating models, *ramping* workload, creating learning and tutoring support and refining documentation. The incremental design process (as described earlier and illustrated in Figure 1) involves the design of models, decisions and results and as appropriate testing and calibrating as the simulation is created - a process that increases model size (number of statements), number of variables, reports and help pages as the design progresses.

MODELLING LANGUAGE

The choice of modelling language impacts readability and testability (Figure 3 shows spreadsheet based and BASIC based models) models.

```
=IF(B33>B84,B84,B33)
```

Figure 3a: Spreadsheet calculation

```
If Sales > Inventory Then  'insufficient inventory to serve demand
   Sales = Inventory       'amount sold limited to available inventory
End If
```

Figure 3b: BASIC language calculation

Figure 3: Language Examples

As revealed in Figure 3b, when sales demand exceeds inventory the actual sales will be the same as inventory (and sales will be lost). When there is sufficient inventory, actual sales will be the same as demand. Besides self documenting the variables, the BASIC example indents the calculations to reveal structure and uses comments (the text following the single quote) to explain the calculation further. In contrast, the Spreadsheet calculation (Figure 3a) does not identify explicitly the variables or explain what the calculation does. And, as shown by the cell numbers, the calculation involves using data from earlier parts of the spreadsheet. When the code is created this is unlikely to be a problem (unless the B33 and B84 cell references are wrong). Readability is crucial when it comes to Code Inspection and it is reasonable to suggest that the BASIC language model is much easier to inspect than the spreadsheet model. The importance of software readability is emphasised by McConnell (2004) who dedicates a whole chapter to the subject.

White Box testing requires stepping through the model and accessing variables during this process. A high level language and integrated development environment like Visual Basic allows one to do this statement by statement, running to pre-defined points in the model or stop (break) when selected variables change or are used. For example, the Visual Basic Integrated Development Environment it is possible to stop and inspect the code and values whenever a variable such as *Inventory* changes. But some languages do not allow one to step through the model exploring calculations.

SOFTWARE STRUCTURE

Kernighan & Plauger (1978) when discussing program structure point out that *"most programs are too big to be comprehended as a single chunk"* and this applies equally to simulation models. Just as software consists of a series of modules (subroutines, function and objects), simulations consist of a series of sub-models (such as price response models, inventory models, income statement models etc.). Kleijnen (1995) when discussing the verification of simulation models posits that a modular structure is good programming practice. Kernighan & Plauger suggest that a modular stucture aids comprehension and this is important because of complexity - especially where is allows the cyclomatic complexity of each sub-model to be at a manageable level (McCabe, 1976; McConnell, 2004). Beyond that, it facilitates adding sub-models, the

associated data and reports incrementally during design and as this is done tested and documented. A further benefit of a modular, sub-model approach is that it allows the designer to build up a library of pre-verified and pre-validated sub-models (objects) that can be used when designing new simulations (Hall, 1996).

Another structural aspect is the extent to which the model (code), data, decision entry and result display are separate. Tjia (2009) advocates as *best practice* separating the model, input (decisions) and output (results) but does not extend this to include data as separate entities (databases).

REFACTORING THE MODELS

Simulation design involves a process of continuous addition and modification that may introduce errors and reduce readability (Khan et al, 2011; Mohammad et al, 2013). Later as the design evolves and the software is changed, added to and tested there may be problems. To guard against these problems the software must be *refactored* (revisited and modified) (Fowler, 1999) to ensure it functions correctly and is readable. Figure 4 shows how the calculations from Figure 3b have been refactored and added to. The variable *Inventory* has been changed to *AvailableInventory, Sales* has been replaced by two variables (*SalesDemand* and *ActualSales)* and additional calculations included to clarify the model. The cost of poor forecasting could have been included in the *actual sales model,* but structurally it was felt that it would be better to create a separate model as part of the business analysis models. But refactoring is not without risk (Spolsky, 2004) because of the way one model interacts with other models. As refactoring involves modifying the software this is a major source of errors (Baisli & Perricone, 1984). For example, in the earlier production model *Inventory* must be changed to *AvailableInventory*, in the earlier marketing model *Sales* must be changed to *SalesDemand* and in the later Income Statement model when calculating *Revenue, Sales* must be changed to *ActualSales.*

```
'actual sales model
If SalesDemand > AvailableInventory Then          'insufficient inventory to serve demand
   ActualSales = AvailableInventory               'sales equal inventory
Else                                              'sufficient inventory to serve demand
   ActualSales = SalesDemand                      'sales equal demand
End If
ClosingInventory = AvailableInventory - ActualSales   'inventory remaining

'analysis of poor forecasting model
LostSales = SalesDemand - Available Inventory         'amount of  business turned away
LostProfit = LostSales * Margin                       'profit lost because demand was not serviced
AverageInventory = (OpeningInventory + ClosingInventory)/2 'average during period
InventoryHoldingCost = AverageInventory * CarryingCost   'economic cost of inventory
```

Figure 4: Refactored Calculations (in *italics*)

DEFENSIVE PROGRAMMING

Defensive programming is intended to ensure the continuing function of a piece of software in spite of unforeseeable usage of the software. The nature of business simulations means that participants have virtually unlimited authority to make decisions and this causes the software to be stressed to extremes - necessitating defensive programming. There are several areas of risk - decision entry, algorithms that are likely to cause exceptions and dynamic instability. It is standard data processing practice to parse input and for business simulations this should extend to decision screening (Hall, 1994) where illegal decisions are identified and rejected and unusual decisions questioned. Run-time exceptions (such as *division by zero*) cause the program to terminate need to be identified and redesigned to prevent failure. Eliminating or minimising the risk of dynamic instability is achieved by constraining feedback loops and delays.

DOCUMENTATION

A key part of software development is documentation (Riggs, 1988)) and, in particular, source code documentation, online help and external documentation. There are two issues with documentation. First, commonly, programmers do not like to document (Spolsky, 2004) and, secondly, documentation of agile software development is often poor (Mohammad et al, 2013).

Source Code documentation reveals and describes the calculations. Kotula (2000) suggests that source code documentation is critical to software development and an *irreplaceable necessity.* Kernighan & Plauger (1978) devote a full

chapter to source code documentation. McConnell (2004) suggest the variable names should be descriptive, reasonably brief and describes the variable unambiguously. As, illustrated in Figures 3b and 4, variable names commonly consist of several words. These are made more readable (Brinkley et al, 1990) by using "medial capitals" (Camel Case or Pascal Case where the start of each word is a capital) or using underscores between words (Snake_Case). Indenting, comments and white space between sub-models improve readability further.

Online help documents the variables and the reports produced. Figure 5 shows an example of the help record used to explain actual sales. In this example the underlined terms are hyper-text links to other help record explaining the terms.

Actual Sales
Actual sales is the number of units actually sold to customers. If there is sufficient inventory then actual sales will be the same as sales demand. But where there is too little inventory, then actual sales will be limited to the amount inventory.

Figure 5: Online Help example

During design, help records facilitate refactoring and when variables are used in other parts of the model. Besides helping during design, online help supports the learners and tutor during use and is a vital aid if redesign is necessary. As illustrated in Figure 1, ideally help records are created as the simulation is created.

External Documentation is the paper based documentation of the simulation models, processing sequence, the decisions and results. In contrast to the source code and on-line documentation that is at a detail level, external documentation is at a high level and descriptive rather that detailing individual algorithms. Figure 6 shows extracts from the external documentation for a stage-gate simulation for middle management of a large engineering design company. This (the Prospector simulation) involves participants searching for project opportunities that matched their capabilities and business objectives, narrow down their choice, tender and negotiate contracts and, finally, fulfil (execute) the contracts

Figure 6a: Key Models	**Figure 6b: Processing Sequence**
Project Ambiguity	Search for Project Opportunities
Pre-Contract Design Impact	Pre-Qualify Project Opportunities
Client Meeting	Prepare Tenders
Project Urgency	Negotiate Contracts
Project Size & Spread	Execute the Projects
Competitor Pricing	
Project Execution	
Cash Flow	

Figure 6: External Documentation example

External documentation serves several purposes. It captures the design as the models are created to facilitate refactoring and integration between models. It can be used as part of the trainers manual and used when the simulation is updated or the models are used by later simulations. The major processing sequence sub-models (Figure 6b) are likely to be defined and documented at the start of the simulation with sub-processes documented as they are added to the design. The key models (Figure 6a) are documented as the simulation design progresses and linked to the processing sequence. This documentation helps ensure correct processing structure and finding models when testing.

VERIFICATION SUPPORT

Methodology, structure and modelling language provide the basis for good quality but beyond this is the question of how you support quality assurance during testing (especially during Black Box testing). This can be accomplished by providing additional reports that provide checks on *intermediate* simulation outputs (Kleijnen, 1995). These explain (reconcile) accounting and operational calculations (Figure 7a), analyse the business (Figure 7b) and validate and help calibrate the black box models (Figures 8). Besides providing data for the quality assurance reports, these additional reports serve to document the model's logic.

In Figure 8, each column shows how individual decisions (price, promotion and product) impact how the customer responds to each. These are these aggregated (Market Response) and together with Nominal Demand determine Sales

Demand. From this the financial impact is calculated in terms of Sales Revenue, Cost of Sales, Promotion Cost and Net Profit. Figure 8 as a whole shows how different prices, promotion and product impact sales, revenue and profit.

Inventory/Sales Reconciliation	
Opening Inventory	100
Actual Production	200
Available Inventory	300
Sales Demand	251
Closing Inventory	49
Actual Sales	251

Figure 7a Reconciliation Report

Forecast Error Cost Analysis	
Average Inventory	74.5
Carrying Cost	.20
Inventory Holding Cost	15
Lost Sales	0
Profit Margin	30
Lost Profit	0

Figure 7b: Business Analysis Report

Figure 7: Reconciliation and Business Analysis Reports

Marketing Decisions Impact								
Price Decision	100	80	120	100	100	100	100	100
Promotion Decision	2500	2500	2500	2000	3000	2500	2500	2500
Product Decision	3	3	3	3	3	2	4	5
Price Response	1.00	1.49	0.72	1.00	1.00	1.00	1.00	1.00
Promotion Response	0.90	0.90	0.90	0.78	0.97	0.90	0.90	0.90
Product Response	0.93	0.93	0.93	0.93	0.93	0.72	1.00	0.91
Market Response	0.84	1.25	0.60	0.72	0.91	0.64	0.90	0.81
Nominal Demand	300	300	300	300	300	300	300	300
Sales Demand	251	374	180	217	272	193	269	244
Price	100	80	120	100	100	100	100	100
Potential Revenue	25200	29920	21600	21700	27200	19300	26900	24400
Cost of Sales	13860	20570	9908	11935	14960	9650	16140	15860
Promotion Cost	2500	2500	2500	2000	3000	2500	2500	2500
Potential Profit	8840	6850	9200	7765	9240	7150	8260	6040

Figure 8: Black Box Validation and Calibration

Hall (2012) while exploring the design of an actual business simulation found that reports to support verification increased the number of reports produced by the simulation nearly three-fold. But, besides supporting verification the extra reports support learning during simulation use. In particular, reconciliations (Figure 7a) allow the trainer to answer questions authoritatively and quickly about how results were calculated. Business Analysis reports (Figure 7b) allow the trainer to identify individual team strengths and weaknesses and decide whether to and how to coach and challenge. Black Box validation reports (Figure 8) reveal how the simulation responds to decisions and where teams are compared identifies relative team strengths and weaknesses and facilitate choosing which teams need coaching and challenge.

CONCLUSIONS

This paper has concentrated on the objective, process aspects of quality assurance needs and ensuring model verification. But, just as business simulations have emotional-engagement aspects there are the subjective aspects of the simulation designer's personality traits and the emotional issues associated with testing and quality management.

Amer (2005) differentiates between analytical and creative thinking and it is arguable that the designing business simulations requires creative thinking and testing and managing quality require analytical thinking. It is possible that these are conflicting personalities and the Myers-Briggs personality inventory suggests that creative and artistic personalities are ENFP and ISFP types - both with the feeling (F) and perceiving (P) - personalities that are different from those of computer software designers. Lyons (1985) and Smith (1989) found that computer software designers have a preponderance of thinking (T) and judging (J) personalities. The Lyons and Smith researches found 81% and 89% respectively were thinking types and 65% and 86% respectively were judging types. Further, Buie (1988) found that ISFP (the Myers-Briggs type associated with artists) was particularly underrepresented with no computer programmers in his sample having this characteristic. These personality differences may impact the assurance of quality. In this context, it would be interesting to research the Myers-Briggs types of simulation designers and users to see if they have creative/artistic or computer software design personalities.

Emotional issues play a part in both testing and quality management. As mentioned, testing is likely to be very time consuming and boring and lead to early termination of the tests or a reduction in diligence and this may cause problems for people with an artistic/creative personality. If there is a problem, this might be mitigated by to delegating testing to people with analytical (TJ) personalities (computer programmers). But, arguably, simulation model testing parallels proof reading literature where for literature the proof reader becomes engaged in the *story* and for the simulation the tester becomes engaged by the *experience.* And, this engagement leads to *flow* (Csikszentmihaly, 2002) where *"experience is so gratifying that people are willing to do it for its own sake"* and thwarts testing. Managing quality during design is even more problematical because, unlike testing, creative needs suggest a need for it to be done by an FP type person. A person who may be unwilling to *step away* from business simulation creation to manage quality. Further, just as *flow* can occur during testing, it can occur during design as the simulation designer sees his or her creation take shape.

Arguably, the special nature of business simulations - the model size, its complexity, who uses it and their emotional engagement means that ensuring and verifying the quality is, perhaps, more difficult than for computer software in general. Quality assurance through testing is unsatisfactory because size and complexity mean that testing for all conditions is unreasonably long and costly. Consequentially, quality assurance is only ensured if quality is managed throughout the design process and the designer is knowledgeable about and uses good software practice taking into account the special nature of business simulations.

REFERENCES

All the ABSEL Proceedings papers can be found in the Bernie Keys Library that is published annually (see http://absel2011.wordpress.com/about/bernie-keyes-library/)

Aldrich, Clark (2009) *The Complete Guide to Simulations and Serious Games* Pfeiffer, San Francisco

Amer, Ayman (2005) *Analytical Thinking* Center for Advancement of Postgraduate Studies and Research in Engineering - Cairo University (CAPSCU), Cairo

Baisli, Victor R. and Barry T. Perricone (1984) Software Errors and Complexity: An Empirical Investigation, *Communications of the ACM* Volume 27, Issue 1, New York

Bellman, Richard, Charles Clark, Cliff Craft, Don O. Malcolm and Franc Ricciardi (1957) On the construction of a multi-stage multi-person Business Game, The RAND Corporation.

Biggs, William D. & Annette L. Halpin (2004) On the Value of Bugs in Simulation Environments, *Developments in Business Simulation and Experiential Learning, Volume 31*

Bots, Pieter & Els van Daalen (2007) Functional Design of games to support natural resource management policy development *Simulation & Gaming Dec 2007, Volume 38*

Brinkley, Dave, Marcia Davis, Dawn Lawrie & Christopher Morrell (2009) To CamelCase or Under_score, *Proceedings of the 2009 IEEE 17th International Conference on Program Comprehension*

Buie, E. A. (1988) Psychological type and job satisfaction in scientific computer professionals, *Journal of Psychological Type, 15*

Cook, Ian (2006) Beware the perils of spreadsheets, *Financial Times* http://www.ft.com/cms/s/2/53faff38-df5e-11da-afe4-0000779e2340.html#axzz2bvPSRaCp (retrieved 8/14/2013)

Csikszentmihaly, Mihaly (2002) *Flow* Random House, London

Dick, Walter, Lou Carey & James O. Cary (2008) *The Systematic Design of Instruction* Pearson, New Jersey

Evans, J & J. W. Dean, jr (2002) *Total Quality: management, organisation and strategy* South-Western, St Paul

Forrester, Jay W. (1958) Industrial Dynamics - A Major Breakthrough for Decision Makers.", in: *Harvard Business Review*, Vol. 36, No. 4, pp. 37–66.

Freeman, D (1996), *How to make spreadsheets error-proof,* Journal of Accountancy, 181 (5), 75

Fowler, Martin (1999) *Refactoring: Improving the design of existing code*. Addison Wesley Longman, Boston Mass.

Gao, Jerry, H._s. Tsao, Ye Wu (2003) *Testing and Quality Assurance for Component-based Software* Artech House, MA, USA

Gill, Geoffrey K, & Chris F. Kemerer (1990) Cyclomatic Complexity Density and Software Maintenance Productivity *IEEE Transactions on Software Engineering* Volume 17 Issue 12, December 1991

Gold, Steven (2003) The Design of a Business Simulation using a System-Dynamics-Based Approach *Developments in Business Simulation and Experiential Learning,,* Volume 30

Gold, Steven C. & Thomas F. Pray (1990) Modelling Demand in Computerized Business Simulations, in *Guide to Business Gaming and Experiential Learning* ABSEL

Goodwin, Jack S. & Stephen G. Franklin Sr. (1994) The Beer Distribution Game: Using Simulations to Teach Systems Thinking, *Journal of Management Development,* 13.8

Goosen, Kenneth R (1981) A Generalised Algorithm for Designing and Developing Business Simulation *Developments in Business Simulation and Experiential Learning,* Volume 8
Goosen, Kenneth R (2007) An Analysis of the Interactions of Firm Demand and Industry Demand in Business Simulations *Developments in Business Simulation and Experiential Learning,* Volume 34
Hall, Jeremy & Benita Cox, (1993) Computerised Management Games: the feedback process and servo-mechanism analogy in *The Simulation & Gaming Yearbook 1993* eds. Fred Percival, Sheila Lodge and Danny Sunders, Kogan Page, London
Hall, Jeremy J. S. B. (1994) Computerised Tutor Support Systems: the tutor's role, needs and tasks, in *The Simulation & Gaming Yearbook Volume 2* Kogan Page London
Hall, Jeremy J. S. B. (1996) Computerized Simulation Design: OOP or oops *The Simulation & Gaming Yearbook Vol 4* Kogan Page, London
Hall, Jeremy J. S. B. (2005) Computer business simulation design: the rock pool method Issues *Developments in Business Simulation and Experiential Learning,* Volume 32
Hall, Jeremy J. S. B. (2007) Computer Business Simulation Design: Novelty and Complexity Issues *Developments in Business Simulation and Experiential Learning,* Volume 34
Hall, Jeremy J. S. B. (2011) *The Art, Science and Craft of Computer Business Simulation Design*, Hall Marketing, London
Hall, Jeremy J. S. B. (2012) Designing the Training Challenge *Developments in Business Simulation and Experiential Learning,* Volume 39
Hansen, W.J. (1978) Measurement of Program Complexity By the Pair (Cyclomatic Number, Operator Count, in *ACM SIGPLAN Notices* 13 (3) March 1978
Howard, Philip (2005) *Managing Spreadsheets,* A White Paper by Bloor Research, Towcester, UK.
IEEE (1990), IEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology
Kernighan, Brian W. & P. J. Plauger (1978) *The Elements of Programming Style* McGraw-Hill Book Company, New York
Khan, Asif Irshad, Rizwan Jameel Qurashi & Usman Alli Khan (2011) A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies *UCSI International Journal of Computer Science,* Volume 8, Issue 4, No 2
Kleijnen, Jack P.C (1995) Verification and validation of simulation models *European Journal of Operational Research 82* Elsevier
Kotler, Peter (1991) *Marketing Management* Prentice Hall, New Jersey
Kotula, Jeffrey (2000) Source Code Documentation: An Engineering Deliverable" in *Proceedings of the Technology of Object-Oriented Languages and Systems*
Law, A.M. & W.D. Kelton (1991) *Simulation Modeling and Analysis 2nd ed.,* McGraw-Hill, New York
Lee, Hau L., V. Padmanabhan & Whang (1997) The Bullwhip Effect in Supply Chains, *Sloan Management Review* Spring 1997
Löwgren, Jonas (1995) *Applying Design Methodology to Software Development*, ACM, New York
Lyons, M.L. (1985) The DP psyhe, *Datamation 31 (16)*
McCabe, Thomas J. (1976) A Complexity Measure. *IEEE Transactions on Software Engineering*: 315
McConnell, Steve (2004) *Code Complete,* Microsoft Press, Redmond
Mohammad, Adel,Tariq Alwada'n, Jafar "M.Ali" Ababneh (2013) Agile Software Methodologies: Strength and Weakness, *International Journal of Engineering and Technology (IJEST)* Volume 5, No 03
Molenda, M (2003) In serch of the elusive ADDIE model *Performance Improvement 42(5)*
Murff, Elizabeth J. Tipton, Richard D. Teach & Robert G. Schwartz (2006) Three-Attribute Interrelationships for Industry-Level Demand Equations *Developments in Business Simulation and Experiential Learning,* Volume 33
Myers, Glenford J., Corey Sandler, Todd M. Thomas and Tom Badgett (2004) *The Art of Software Testing* John Wiley & Sons, Inc. New Jersey
Panko, R and Halverson, R. (1996) Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks *Proceedings of the Twenty-Ninth Hawaii International Conference on Systems Sciences*, Maui, HA, January 1996
Panko, Raymond R. (2005) Spreadsheet Errors: What We Know, What We Think We Can Do, *Proceedings of the Spreadsheet Risk Conference*, Greenwich, England, July 2000
QSM - Function Point Languages Table Version 5.0 2013 - http://www.qsm.com/resources/function-point-languages-table (retrieved 19/7/2013)
Rajalingham, K, Chadwick, D. and Knight, B. (2000) Classification of Spreadsheet Errors*, British Computer Society (BCS) Computer Audit Specialist Group (CASG) Journal,* Vol 10, No 4 (Autumn 2000)
Riggs, Robert (1988) Computer Systems Maintenance. *Techniques of Program and System Maintenance*. QED Information Sciences,
Shepperd, M. (1988) A critique of cyclomatic complexity as a software metric, *Software Engineering Journal* 3 (2) (March 1988)

Smith, D. C. (1989) The personality of the systems analyst: and investigation, *ACM Computer Personnel 12 (2)*
Spolsky, Joel (2004) *Joel on Software* Apress, Berkeley, CA
Teach, Richard D. & Robert G. Schwartz (2000) Introducing Cross-Elasticities in Demand Algorithms *Developments in Business Simulation and Experiential Learning,* Volume 27
Thavikulwat, Precha (2004) The architecture of computerized business gaming simulations *Simulation & Gaming* Volume 38 Number 2 June 2007 SAGE Publications
Tjia, John S. (2009) *Building Financial Models* McGraw Hill, New York

THE SIMULATIONS

This list of simulations are the ones used for the cyclomatic complexity analysis and referenced in the paper. They show the *original* design date. However all have been updated since then.

Hall, Jeremy J. S. B., Business Focus, Hall Marketing, 2000

Hall, Jeremy J. S. B., Distrain, Hall Marketing, 2004

Hall, Jeremy J. S. B., Executive Challenge, Hall Marketing, 1996

Hall, Jeremy J. S. B., Foundation Challenge, Hall Marketing, 2002

Hall, Jeremy J. S. B., Global Operations, Hall Marketing, 1981

Hall, Jeremy J. S. B., Management Challenge, Hall Marketing, 1986

Hall, Jeremy J. S. B., Management Experience, Hall Marketing, 1976

Hall, Jeremy J. S. B., Operations, Hall Marketing, 1981

Hall, Jeremy J. S. B., Product Launch, Hall Marketing, 1978

Hall, Jeremy J. S. B., Professionals Challenge, Hall Marketing, 2013

Hall, Jeremy J. S. B., Prospector, Hall Marketing, 2005

Hall, Jeremy J. S. B., Retail Challenge, Hall Marketing. 1987

Hall, Jeremy J. S. B., Sales Calls, Hall Marketing, 1983

Hall, Jeremy J. S. B., Sales Mix, Hall Marketing, 1983

Hall, Jeremy J. S. B., Service Challenge, Hall Marketing, 1989

Hall, Jeremy J. S. B., Service Launch, Hall Marketing, 2008

Hall, Jeremy J. S. B., SMART, Hall Marketing, 1987

Hall, Jeremy J. S. B., SMITE, Hall Marketing, 1984

Hall, Jeremy J. S. B., Teamskill, Hall Marketing, 1971

Hall, Jeremy J. S. B., Training Challenge, Hall Marketing, 2011