

Requirements Evaluation Using Behavior Trees - Findings from Industry

Daniel Powell
danielpowell.com.au
daniel@danielpowell.com.au

Abstract

Requirements specifications are usually evaluated using traditional review, inspection and walkthrough techniques, varying in both rigor and repeatability. These methods, while beneficial, are difficult and expensive to apply in the evaluation of requirements for large and complex systems (requirements documents comprising thousands of requirements over hundreds of pages).

The Behavior Tree method suggests that a process of individual requirement translation to a formal representation followed by integration of these formal representations provides a means to deal with the complexity inherent in the specifications of real systems. We propose that the reading technique defined by the Behavior Tree method affords requirements evaluations that are more repeatable, more effective, and higher yield than do traditional formal inspection reading techniques.

This paper provides quantitative support to these claims, by presenting and analysing the data collected during the behavior tree analysis and evaluation of the requirements of five large (each between approximately 800-1200 requirements) and complex systems. We compare a number of key metrics to those obtained from traditional formal requirements reviews and inspections.

This paper will demonstrate that the Behavior Tree method provides a major defect detection rate significantly higher than traditional review and inspection rates while at the same time producing a model that is useful as a tool to support validation with domain experts. Although alluded to, this paper does not present any evidence supporting the use of the resulting models for refinement to an implementable or acquirable solution.

1. Introduction

Useful processes, that are independently repeatable, are utilised in all branches of science and traditional engineering disciplines but seldom in software engineering. This is

particularly so with processes used for the detection of defects in software systems requirements.

The business case for detecting defects early in the life cycle is a strong one:

- reworking requirements defects on most software development projects costs between 40% [3] and 80% [6] of total project effort.
- requirements defects may cost between 10 to 200 times as much if detected in a fielded systems or 10 times as much if detected during testing compared to detection at the requirements stage [3].
- as much as 60% of all defects in a systems lifetime originate from deficient requirements [1]

While effective, traditional reviews and inspections vary in rigor and in repeatability, and therefore, in effectiveness. The Behavior Tree method, reported on in this paper, formalises the reading technique employed in the evaluation of requirements. This method improves both the rigor and repeatability of requirements evaluation, consistently leading to higher yield evaluations. One organisation involved in the trial evaluations, consistently found that the Behavior Tree method identified serious defects not identified by their existing review processes in about 10-15% of their requirements.

In addition this paper reports data that indicates the method tree method achieves a defect detection rate between 2 to 4 times that of traditional review techniques [5]. That is, the Behavior Tree method is not only more effective, but more efficient than traditional requirements review techniques.

The remainder of this paper is organised as follows. Section 2 provides a description of the Behavior Tree method. In Section 3 we discuss and analyse the results of applying the Behavior Tree method to the specifications of five separate large and complex industry systems. In section 4 we conclude this paper by discussing a path leading o a capability to employ behavior trees.

2. The Behavior Tree Method

The Behavior Tree Notation captures in a simple tree-like form of composed component-states what usually needs to be expressed in a mix of other notations. Behavior is expressed in terms of components realizing states, augmented by the logic and graphic forms of conventions found in programming languages to support composition, events, control-flow, data-flow, and threads. Behavior trees are equally suited to capture behavior expressed in the natural language representation of functional requirements as to provide an abstract graphical representation of behavior expressed in a program. To use David Harel's metaphor, Behavior Trees represent a lifting up of behavior expressed in programming languages to a higher level of abstraction.

Definition: A Behavior Tree is a formal, tree-like graphical form that represents behavior of individual or networks of entities which realize or change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control.

To support the implementation of software intensive systems we must capture, first in a formal specification of the requirements, then in the design, and finally in the software; the actions, events, decisions, and/or logic obligations, and constraints expressed in the original natural language requirements for a system. Behavior trees do this. They provide a direct and clearly traceable relationship between what is expressed in the natural language representation and its formal specification. Translation is carried out on a sentence-by-sentence, word-by-word basis. Figure 1 shows a sample translation to a behavior tree. Components are in **bold** and *states, conditions* and *events* are in *italics*. A brief summary of key elements of the notation is given in Figure 2, (see web-site <http://www.sqi.gu.edu.au/gse/fordetails>).

Behavior trees provide a direct and clearly traceable relationship between what is expressed in the natural language representation and its formal Behavior Tree counterpart. The Behavior Tree method allows the engineer to manage complexity and scale in the construction of a Behavior Tree model which is "built out of" its requirements (this implies the weaker property that the model must satisfy the requirements). The Behavior Tree method achieves this through a process of requirements translation followed by incremental integration.

Requirements Translation

Requirements translation is the first formal step in the method and the first place where we have a chance to

Behavior

When a **car** is at the entrance if the **gate** is open the **car** may proceed, otherwise if the **gate** is closed, when and if the **driver** presses the **button** the gate will open and then the **car** may proceed.

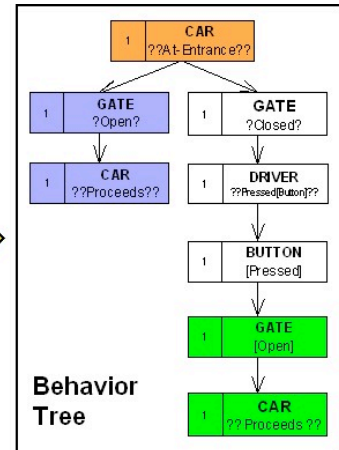


Figure 1. Translation of natural language to a Behavior Tree

Component-State Label	Semantics
	Indicates that the component has realized the particular internal state and then passes control to its output.
	Indicates that the component will assign a value to one of its attributes.
	Indicates that the component will only pass control if if-state is TRUE
	Indicates that the component will only pass control when AND if the event WHEN-state happens after reaching this component-state.
	Indicates that the component will only pass control when the event WHEN-state happens OR has happened prior to reaching this component-state.
	Indicates that when the component has realized the state it will pass the data to the component that receives the flow.
	Indicates that when the component has realized the state it will have received the data from the component that sends the flow.
	The system component, System-Name realizes the state "State" and then passes control to its output.

Figure 2. Behavior Tree Notation, Key Elements

uncover requirements defects. Its purpose is to translate each natural language functional requirement, one at a time, into one or more behaviour trees called a requirements behavior tree (RBT). Translation identifies the components (including actors and users), the states they realise (including attribute assignments), the events and decisions/constraints that they are associated with, the data components exchange, and the causal, logical and temporal dependencies associated with component interactions. As the translation is carried out on a requirement-by-requirement basis independent of other requirements, this effort does not tax human short-term memory, regardless of the scale and complexity of a specification.

Translation Defect Detection

During initial translation of functional requirements to behaviour trees there are five principal types of defects that we encounter:

- **Aliases** exist where different words are used to describe a particular entity, state, action, event, etc... For example, in one place the requirements might refer to the `Uplink-ID` while in another place it is referred to as the `Uplink-Site-ID`. It is necessary to maintain a vocabulary of component names and a vocabulary of states associated with each component to maximize our chances of detecting aliases. A good requirements specification, one that reduces the chance of misinterpretation, will possess a single, consistent vocabulary.
- **Ambiguities** are detected where not enough context has been provided to allow us to distinguish among more than one possible interpretation of the behaviour described. Ambiguity is often a result of loose language in a requirement. Common examples involve the use of terms like `within`, `satisfactory`, ... Ambiguity defects are often identified during translation when the formal language does not tolerate a translation of the ambiguous requirements. Despite this, unfortunately, there is no guarantee that an engineer will always recognize an ambiguity when doing a translation this obviously impacts on our chances of achieving repeatability when doing translations.
- **Incompleteness** can be identified during translation as missing, implied and/or alternate behavior. The behavior tree method does not add any information to a specification unless behavior is missing. Many errors of omission involve tacit and semi-tacit knowledge not possessed by the translating engineer. These types of incompleteness problems are not found during translation (or integration), but often during a walkthrough of the resulting model with domain experts (i.e. during validation).

- **Inaccuracy** is identified as incorrect causal, logical and temporal attribution. Inaccurate atomic statements, usually specified values or ranges, may also be inaccurate. These types of defects are often identified during modelling, however, this identification is dependent on possession of domain expertise and, therefore, not repeatable.
- **Inconsistency** is detected during translation if a single requirement statement is inconsistent within itself.

Requirements Integration

Once requirements are translated to individual requirements behavior trees (RBTs), they can be integrated by the precondition and interaction axioms [2]. In practice, it most often involves locating where (if at all) the component-state root node of one behaviour tree occurs in the other tree, and grafting the two trees together at that point. This process generalises when we need to integrate N behaviour trees. We only ever attempt to integrate two behaviour trees at a time either two RBTs, an RBT with a DBT or two partial DBTs. As for translation, integration can be performed without regard for order, again facilitating the engineer's ability to handle scale and complexity by allowing concentration on just the requirements trees being integrated.

Integration Defect Detection

Attempts at requirements integration using the Behavior Tree method uncover many defects with the requirements early in the life cycle. Traditional review methods struggle to find such issues as they depend on a human reader reading a document in order, limited by short-term memory.

- **Aliases** are further detected during integration as requirements are integrated in context. Often when one component is given two different names it becomes apparent during integration (each has incomplete behavior but complementary behavior - that is together they are complete).
- **Ambiguities** are often detected as an incompleteness in contextual information during integration. That is ambiguous statements make it difficult to properly model preconditions, which in turn lead to an inability to integrate behavior.
- **Incompleteness** is usually associated with either incomplete pre and post conditions making integration difficult or impossible, with incomplete sets of events for triggering a behavior, or with incomplete sets of conditions.
- **Inaccuracy** defect detection is facilitated as behavior is placed in context.

	Project 1	Project 2	Project 3	Project 4	Project 5			
Measures								
Total number of pages analysed	106	90	69	90	110			
Total number of requirements analysed	1272	920	950	785	1000			
Total Number of Major Defects Identified	172	109	131	110	156	% of Total		
Incompleteness	101	72	87	74	84	61.7%		
Inconsistency	14	8	8	2	5	5.5%		
Ambiguity	40	23	30	26	30	22.0%		
Redundancy	4	5	4	2	34	7.2%		
Inaccuracy	13	1	2	6	3	3.7%		
Time (hours)								
Plan Time	10.00	12	4	3	8			
Analysis / Modeling Time	83	82	76	79	90			
Reporting Time	16	16	10	6	5			
Consultation Time	8	4	4	2	5			
Total Effort	117	114	94	90	108			
						Average	Pop Var	Pop SD
Pages / Hour	0.91	0.79	0.73	1.00	1.02	0.89	0.013	0.11
Requirements / Hour	10.87	8.07	10.11	8.72	9.26	9.41	0.983	0.99
Defects Identified / Hour	1.47	0.96	1.39	1.22	1.44	1.30	0.037	0.19
Defects / Page	1.62	1.21	1.90	1.22	1.42	1.47	0.068	0.26
Defects / Requirement	0.14	0.12	0.14	0.14	0.16	0.14	0.000	0.01

Figure 3. Summarised Trial Project Data

- **Inconsistency** is detected during integration as a formal integration problem. That is, attempting to integrate two or more inconsistent RBTs into the one DBT would result in contradictory behavior. Inconsistency defects can be model checked.
- **Redundancy** is also detectable as an integration problem. Redundancy is to be considered a serious defect as it has an impact not only on understanding but on change management (changes, if not applied to all redundant requirements, may result in inconsistency).

Many defects with requirements can be discovered only by creating an integrated view, because examining requirements individually gives us no clue that there is a problem.

The complete integrated DBT represents the entire system as defined by the specification being analysed. This model, showing one integrated view of all behavior and contextual information proves a valuable tool for validation with domain experts. As all behavior is represented in context, domain experts readily identify errors of omission, inappropriately detailed areas of specification and incorrect preconditions and postconditions.

Component models, and component networks can be automatically projected from an integrated DBT. These representations make it easier to identify errors of omissions with single component specifications. For example, projecting a user behavior tree (a tree showing only user behavior - equivalent to a state machine for the user component) often highlights that user behavior has been inappropriately assigned to certain users or omitted entirely.

3. Case Study Findings

In this section, data is presented supporting the claim that the Behavior Tree method is an efficient and effective method of early life cycle defect detection, and that it greatly outperforms traditional requirements reviews and inspections.

The Data: The data reported on in this section is based on defence, transport, and finance and banking systems.

The specifications evaluated during these trials had sizes between approximately 800 and 1250 requirements. All were specified using traditional "shall" statements¹

The data was collected over a period of eight months. Each of the evaluations was conducted by the same individual, non-domain expert, using a technology non-specific tool; OmniGraffle (similar to Smartdraw, Visio, etc...).

All defects included in the reported defect data have been verified by the respective domain experts on the trial projects. Only major defects are reported.

The time-based metrics are based on the total evaluation time, including times for analysis and modeling, defect logging, consultation with domain experts and production of summary reports.

In each evaluation, there was no prior walkthrough of the reviewed document or other related artefacts and, in most, a single meeting toward the end of analysis to clarify issues. In most cases, the majority of queries raised were considered defects by domain experts (i.e. dependent on knowl-

¹ Data collected from small projects using use-case specification indicate an even higher defect detection rate - twice again of that reported for these projects.

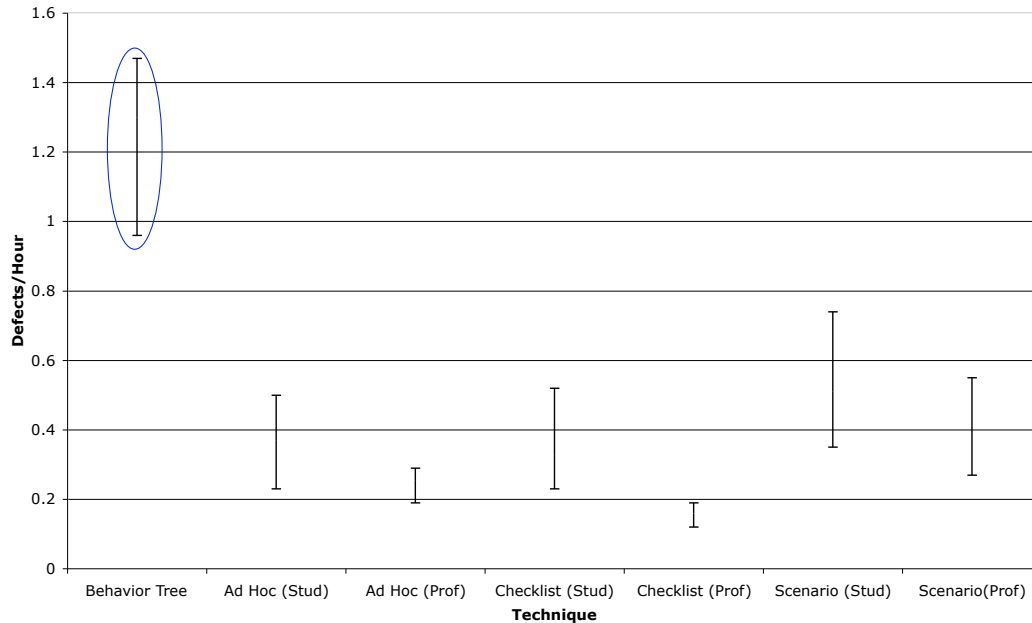


Figure 4. Defect Detection Rate: Comparison of Techniques

edge that should have been stated) and were subsequently listed as defects (usually classified as incompleteness).

At the time of reporting this data, a number of queries remained outstanding. These queries have not been reported and possibly indicate a higher defect density (and detection rate) than that reported.

The defect detection and review rates reported were not uniform. Certain sections of the specifications were more complex, more defect dense, or both. These sections took significantly longer to analyse. The data has not been decomposed to this level of reporting (however that would be useful for future estimation). The respective percentages of total effort of the Behavior Tree stages of translation and integration were also non-uniform.

One of the primary advantages of the Behavior Tree method over traditional evaluation techniques is the construction of an integrated model of the entire specification. This model is very useful for validation as it puts requirements in their context, regardless of their location in the source specification. Although the analyses were conducted by a non-domain expert, the great majority of defects were identified during the analysis (this may be an indicator of repeatability). The majority of issues identified during validation of the model by domain experts related to errors of omission.

The data from the five trial projects is reported in figure 3.

Analysis: Figure 3 presents measures of time and defects and key metrics including defect detection rate (defects identified / hour) and review rates. These metrics are used

for comparison against existing, published industry data in the section below.

Although the population size is small, the data is consistent, indicating the repeatability of the method.

In addition to reporting rates, we report defect densities. This data is not analysed in this paper, however, its consistency (particularly the requirements-based metric) implies that it may be useful as an indicator of baseline readiness.

Below we compare key metrics with existing published and unpublished data.

Comparisons with Traditional Requirements Evaluation: Comparisons are made with the data from [5]. The Porter data is based on experiments with both student inspection teams and professional teams applying ad hoc, checklist-based and scenario-based reading techniques. The experiment derives data from a much larger sample than what is reported on here, however, it is based on a small set of requirements. Anecdotal evidence from experience with these techniques indicates that they fail to scale to the evaluation of large sets of requirements, such as those used in these trials.

The method yielded a large number of defects at an average defect detection rate approximately 2 to 3 times that of the ad-hoc, checklist-based, and scenario review techniques in [5]. This comparison is illustrated in Figure 4.

This efficiency is achieved despite the fact that the review rate (including analysis, modelling and reporting) is approximately half the traditional review rate quoted in the Team Software Process (TSP) guidelines (approx. 2

pages/hour) [4]. It must be noted, however, that in addition to the defect log produced by traditional techniques, the Behavior Tree method yields a formal and integrated model that is suitable for validation of the requirements via walk-through (and with current Behavior Tree research, animation and simulation), refinement, change management, and verification.

The method finds significantly more defects than traditional techniques. One organisation involved in the trial evaluations, consistently found that the Behavior Tree method identified serious defects not identified by their existing (mature) review processes in about 10-15% of their requirements. These defects tended to be related to incompleteness and inconsistency. These are the type of defect that become expensive to identify and repair at later stages of the life cycle.

Anecdotal evidence also indicates that the Behavior Tree method, by formalising the reading process, focusses more on major defect detection and does not encourage minor defect reporting (although they are often identified during translation).

4. Conclusion and Further Conjecture

Although based on a small trial population, the data reported on in this paper indicates that the Behavior Tree method provides a high yield requirements evaluation technique that is both more effective and more efficient than traditional requirements evaluation techniques. The technique is also scalable to large specifications as demonstrated by these evaluations.

Based on these trials, past experience with smaller scale systems, and the formalities of the method, we conjecture that teams of engineers would be able to independently and concurrently evaluate different sections of a large requirements specification, greatly decreasing evaluation duration. Such a process requires maintenance of a common vocabulary. Tools supporting the construction of a common vocabulary and interactive integration are currently under construction by the Software Quality Institute at Griffith University.

Adoption: As the method is not reliant on technology, its adoption is relatively simple. The SQI at Griffith University is the first point of contact for any organisation wishing to train in the method.

The Behavior Tree method is a rigorous evaluation technique that is proving to be efficient and effective in evaluating even large and complex sets of requirements. It is satisfying the need for high yield evaluation techniques for identifying, potentially very expensive, defects at the critical early life cycle stages.

Acknowledgement

These industry studies were carried out while Dr Powell was working as an independent consultant for the ARC Centre for Complex Systems (Griffith University and University of Queensland nodes) and the Software Quality Institute (Griffith University). Part of the funding for this work was provided by industry funds received by the Centre and the Institute.

References

- [1] D. M. Berry. Formal methods: the very idea - some thoughts about why they work when they work. *Sci. Comput. Program.*, 42(1):11–27, 2002.
- [2] R. Dromey. From requirements to design: Formalizing the key steps (invited keynote address). In *SEFM-2003, IEEE International Conference on Software Engineering and Formal Methods*, pages 2–11, Sept. 2003.
- [3] D. Firesmith. The business case for requirements engineering. In *RE'2003*, 2003.
- [4] W. Humphreys. The team software process (tsp). *Technical Report CMU/SEI-2000-TR-023 ESC-TR-2000-023*, 2000.
- [5] A. Porter and L. Votta. Comparing detection methods for software requirements inspections: A replication using professional subjects. *Empirical Software Engineering: An International Journal*, 3(4):355–379, December 1998.
- [6] K. E. Wiegers. Inspecting requirements,. *StickyMinds.com Weekly Column*, 30 July 2001.