Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

On the Testability of BDI Agent Systems (Extended Abstract)*

Michael Winikoff and **Stephen Cranefield**
University of Otago
Dunedin, New Zealand
{michael.winikoff, stephen.cranefield}@otago.ac.nz

Abstract

Before deploying a software system we need to assure ourselves (and stakeholders) that the system will behave correctly. This assurance is usually done by testing the system. However, it is intuitively obvious that adaptive systems, including agent-based systems, can exhibit complex behaviour, and are thus harder to test. In this paper we examine this "obvious intuition" in the case of Belief-Desire-Intention (BDI) agents, by analysing the number of paths through BDI goal-plan trees. Our analysis confirms quantitatively that BDI agents are hard to test, sheds light on the role of different parameters, and highlights the enormous difference made by failure handling.

1 Introduction

As agent-based systems [Wooldridge, 2002] are increasingly deployed (e.g. [Munroe *et al.*, 2006; Benfield *et al.*, 2006; Müller and Fischer, 2014]), the issue of *assurance* rears its head. Before deploying a system, we need to convince those who will rely on the system (or those who will be responsible if it fails) that the system will, in fact, work. Traditionally, this assurance is done through testing. However, there is a generally held intuition that agent systems exhibit complex behaviour, which makes them hard to test. This paper tests this "obvious intuition", focussing on Belief-Desire-Intention (BDI) agents [Rao and Georgeff, 1991; Bratman, 1987; Bratman *et al.*, 1988].

The difficulty of testing a BDI agent program can be reduced to test set adequacy: an agent program P is easy to test if and only if there exists a test set T which is *adequate* for testing P with respect to the selected test adequacy criterion, where T is not infeasibly large. There are many criteria that can be used to assess whether a given set of tests is adequate (for a recent overview, see Mathur [2008]). Given that we are interested in assessing the difficulty of testing a given program, we are clearly looking at "white box" testing. Furthermore, we will be working with abstract "goal-plan trees" rather than detailed programs (see Section 2). This means that we need to consider control-flow based metrics, rather than data-flow, since an abstract goal-plan tree does not contain data-flow information. Specifically, we choose the "all paths" criterion[1], using the standard approach for dealing with loops by bounding them [Zhu *et al.*, 1997, p. 375].

We therefore explore the intuition that "agent systems are hard to test because they exhibit complex behaviour", by deriving the number of paths through a BDI program as a function of various parameters (e.g. the number of applicable plans per goal). This naturally leads us also to consider how the number of paths is affected by these various parameters. As might be expected, we show that the intuition that "agent systems are hard to test" is correct, i.e. that agent systems have a very large number of paths. We also show that BDI agents are *harder* to test than procedural programs, by showing that the number of paths through a BDI program is much larger than the number of paths through a similarly-sized procedural program.

Although there has recently been increasing interest in testing agent systems [Zhang *et al.*, 2009; Ekinci *et al.*, 2009; Gomez-Sanz *et al.*, 2009; Nguyen *et al.*, 2009; Padgham *et al.*, 2013], there has been surprisingly little work on determining the feasibility of testing agent systems in the first place. Padgham and Winikoff [2004] analyse the number of successful executions of a BDI agent's goal-plan tree (defined in Section 2), but they do not consider failure or failure handling in their analysis, nor do they consider testability implications.

There are similarities between Hierarchical Task Network (HTN) planning [Erol *et al.*, 1994; 1996] and BDI execution [de Silva and Padgham, 2004]. However, the problem of HTN planning is about *finding* a plan, whereas BDI planning involves interleaving plan search and plan execution [Sardina and Padgham, 2011, p. 45], so the complexity results for HTN planning do not apply: it is a different problem.

The contribution of this work is threefold. Firstly, it confirms the intuition that BDI programs are hard to test. Secondly, it does so by *quantifying* the number of paths, as a function of parameters of the BDI program. Thirdly, we find some surprising results about how the parameters influence the number of paths.

*This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Winikoff and Cranefield, 2014].

[1]Note that the "all paths" criterion considers the parts of the program that were traversed, not the values of variables. So for instance the (trivial) program $x := x \times x$ has a single path, but many traces ($x = 1, 2, 3, \ldots$).

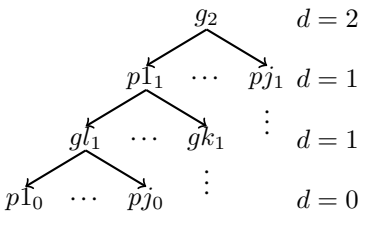
2 Number of Paths Analysis for BDI Agents

BDI plan execution is a dynamic process that progressively executes actions as goals are posted. However, in order to more easily analyse this process, we instead view BDI execution as a nondeterministic data transformation from a *goal-plan tree* in to a sequence of action executions. The goals and plans can be visualised as a tree where each goal has as children the plan instances that are applicable to it, and each plan instance[2] has as children the sub-goals that it posts. This *goal-plan tree* is an "and-or" tree: each goal is realised by one of its plan instances ("or") and each plan instance needs all of its sub-goals to be achieved ("and").

We now consider how many paths there are through a goal-plan tree that is being used by a BDI agent to realise a goal[3] using that tree. Since we view BDI execution as transforming a goal-plan tree into action traces, we proceed by deriving formulae that compute the number of paths, both successful and unsuccessful, for a given goal-plan tree.

We make the following *uniformity* assumptions regarding the form of the goal-plan tree that allow us to perform the analysis: (i) all subtrees of a goal or plan node have the same structure; (ii) we assume that all plan instances at depth $d > 0$ (see below) have k sub-goals; and (iii) we assume that all goals have j applicable plan instances[4].

We define the *depth* of a goal-plan tree as the number of layers of goal nodes it contains. A goal-plan tree of depth 0 is a plan with no sub-goals, while a goal-plan tree of depth $d > 0$ is either a plan node with children that are goal nodes at depth d or a goal node with children that are plan nodes at depth $d-1$ (see example tree).



Terminology: Our uniformity assumptions mean that the structure of the subtree rooted at a goal or plan node is determined solely by its depth, and we can therefore denote a goal or plan node at depth d as g_d or p_d (respectively). We use $n^{\checkmark}(x_d)$ to denote the number of *successful* execution paths of a goal-plan tree of depth d rooted at x (where x is either a goal g or a plan p). Where specifying d is not important we will sometimes elide it, writing $n^{\checkmark}(x)$. Similarly, we use $n^{\times}(x_d)$ to denote the number of *unsuccessful* execution paths of a goal-plan tree of depth d with root x (either g or p).

2.1 Base Case: Successful Executions

We begin by calculating the number of *successful* paths through a goal-plan tree in the absence of failure (and of failure handling). This analysis follows that of Padgham & Winikoff [2004, pp. 17–19].

Roughly speaking, the number of ways a goal can be achieved is the *sum* of the number of ways in which its children can be achieved (since the children represent alternatives, i.e. the goal is represented by an "or" node). On the other hand, the number of ways a plan can be achieved is the *product* of the number of ways in which its children can be achieved, since the children must all be achieved.

Given a tree with root g (a goal), assume that each of its j children can be achieved in n different ways[5]; then, because we select one of the children, the number of ways in which g can be achieved is jn (i.e. $n^{\checkmark}(g_d) = j\, n^{\checkmark}(p_{d-1})$). Similarly, for a tree with root p (a plan), assume that each of its k children can be achieved in n different ways, then, because we execute all of its children, the number of ways in which p can be executed is $n \cdots n$, or n^k, i.e. $n^{\checkmark}(p_d) = n^{\checkmark}(g_d)^k$. A plan with no children (i.e. at depth 0) can be executed (successfully) in exactly one way ($n^{\checkmark}(p_0) = 1$). This can be simplified (for $k > 1$) to the following (and if $k = 1$ then $n^{\checkmark}(g_d) = n^{\checkmark}(p_d) = j^d$):

$$n^{\checkmark}(g_d) = j^{(k^d-1)/(k-1)}$$
$$n^{\checkmark}(p_d) = j^{k\,(k^d-1)/(k-1)}$$

2.2 Adding Failure

We now extend the analysis to include failure, and determine the number of *unsuccessful* executions, i.e. executions that result in failure of the attempt to achieve the top-level goal. For the moment we assume that there is no failure handing (we add failure handling in Section 2.3).

In order to determine the number of failed executions we have to know where failure can occur. In BDI systems there are two places where failure occurs: when a goal has no applicable plan instances, and when an action (within an applicable plan instance) fails. However, our uniformity assumption means that we do not address the former case—it is assumed that a goal will always have j instances of applicable plans.

In order to model the latter case we need to extend our model of plans to encompass actions. We assume there are ℓ actions before, after, and between the sub-goals in a plan, and that a plan with no sub-goals is considered to consist of ℓ actions. For example, for $\ell = 1$ a plan at depth $d > 0$ might have a body of the form $a1; ga; a2; gb; a3$ where ai are actions, ga and gb are sub-goals, and ";" denotes sequential execution.

A plan at depth 0 can fail at each of the ℓ actions, so there are ℓ distinct failing paths ($n^{\times}(p_0) = \ell$). A plan at depth $d > 0$ can fail in the initial ℓ actions, or ("+") the first subgoal can fail ($n^{\times}(g_d)$), or it can succeed and then one of the following ℓ actions fails ($n^{\checkmark}(g_d)\,\ell$), etc. This yields the following definitions for the number of *unsuccessful* executions of a goal-plan tree, *without* failure handling[6].

[2] We assume that a plan body is a sequence of steps, where each step is either an action (which can succeed or fail) or a sub-goal.

[3] We focus on a single goal in our analysis.

[4] In our analysis we make a simplifying assumption. Instead of modelling the instantiation of plans to plan instances, we assume that the goal-plan tree contains applicable plan instances.

[5] Because the tree is assumed to be uniform, all of the children can be achieved in the same number of ways, and are thus interchangeable in the analysis, allowing us to write $j\,n$ rather than $n_1 + \ldots + n_j$.

[6] The equation for $n^{\times}(g_d)$ is derived using the same reasoning as in the previous section: a single plan is selected and executed, and there are j plans.

$$
\begin{aligned}
n^{\text{✗}}(g_d) &= j\, n^{\text{✗}}(p_{d-1}) \\
n^{\text{✗}}(p_0) &= \ell \\
n^{\text{✗}}(p_d) &= \ell + (n^{\text{✗}}(g_d) + n^{\text{✓}}(g_d)\,\ell)\,(1 + \cdots + n^{\text{✓}}(g_d)^{k-1})) \\
&= \ell + (n^{\text{✗}}(g_d) + \ell\, n^{\text{✓}}(g_d))\ \frac{n^{\text{✓}}(g_d)^k - 1}{n^{\text{✓}}(g_d) - 1} \\
&\quad \text{(for } d > 0 \text{ and } n^{\text{✓}}(g_d) > 1)
\end{aligned}
$$

2.3 Adding Failure Handling

We now consider how the introduction of a failure-handling mechanism affects the analysis. A common means of dealing with failure in BDI systems is to respond to the failure of a plan by trying an alternative applicable plan for the (sub-)goal that triggered that plan. For example, suppose that a goal g has three applicable plans $p1$, $p2$ and $p3$, that $p1$ is selected, and that it fails. Then the failure-handling mechanism will respond by selecting $p2$ or $p3$ and executing it. Assume that $p3$ is selected. Then if $p3$ fails, the last remaining plan ($p2$) is used, and if it too fails, then the goal is deemed to have failed.

The result of this is that, as we might hope, it is *harder* to fail: the only way a goal execution can fail is if *all* of the applicable plans are tried and *each* of them fails.

The number of executions can then be computed as follows: if a goal g_d has j applicable plan instances, each having $n^{\text{✗}}(p_{d-1})$ unsuccessful executions, then we have $n^{\text{✗}}(p_{d-1})^j$ unsuccessful executions of all of these plans in sequence. Since the plans can be selected in any order we multiply this by $j!$ yielding $n^{\text{✗}}(g_d) = j!\, n^{\text{✗}}(p_{d-1})^j$. The number of ways in which a plan can fail is still defined by the same equation—because failure handling happens at the level of goals—but where $n^{\text{✗}}(g)$ refers to the new definition:

$$
\begin{aligned}
n^{\text{✗}}(g_d) &= j!\, n^{\text{✗}}(p_{d-1})^j \\
n^{\text{✗}}(p_0) &= \ell \\
n^{\text{✗}}(p_d) &= \ell + (n^{\text{✗}}(g_d) + \ell\, n^{\text{✓}}(g_d))\ \frac{n^{\text{✓}}(g_d)^k - 1}{n^{\text{✓}}(g_d) - 1} \\
&\quad \text{(for } d > 0 \text{ and } n^{\text{✓}}(g_d) > 1)
\end{aligned}
$$

Turning now to the number of *successful* executions (i.e. $n^{\text{✓}}(x)$) we observe that the effect of adding failure handling is to *convert failures to successes*, i.e. an execution that would otherwise be unsuccessful is extended into a longer execution that may succeed.

Consider a simple case: a depth 1 tree consisting of a goal g with three children: $p1$, $p2$, and $p3$. Previously the successful executions corresponded to each of the pi (i.e. select a pi and execute it). However, with failure handling, we now have the following additional successful executions: $p1$ fails, then $p2$ is executed successfully; or $p1$ fails, $p2$ is then executed and fails, and then $p3$ is executed and succeeds. This leads to a definition of the form

$$n^{\text{✓}}(g) = n^{\text{✓}}(p1) + n^{\text{✗}}(p1)\, n^{\text{✓}}(p2) + n^{\text{✗}}(p1)\, n^{\text{✗}}(p2)\, n^{\text{✓}}(p3)$$

However, we need to account for different orderings of the plans. For instance, the case where the first selected plan succeeds (corresponding to the first term, $n^{\text{✓}}(p1)$) in fact applies for each of the j plans, so the first term, including different orderings, is $j\, n^{\text{✓}}(p)$. Similarly, the second term ($n^{\text{✗}}(p1)\, n^{\text{✓}}(p2)$), corresponding to the case where the initially selected plan fails but the next plan selected succeeds, in fact applies for j initial plans, and then for $j-1$ next plans, yielding $j\,(j-1)\, n^{\text{✗}}(p)\, n^{\text{✓}}(p)$.

Continuing this process and then generalising yields the following equations (again, since failure handling is done at the goal level, the equation for plans is the same as in Section 2.1):

$$
\begin{aligned}
n^{\text{✓}}(g_d) &= \sum_{i=1}^{j} n^{\text{✗}}(p_{d-1})^{i-1}\, n^{\text{✓}}(p_{d-1})\ \frac{j!}{(j-i)!} \\
n^{\text{✓}}(p_0) &= 1 \\
n^{\text{✓}}(p_d) &= n^{\text{✓}}(g_d)^k \ \text{(for } d > 0\text{)}
\end{aligned}
$$

Table 1 makes the various equations developed so far concrete by showing illustrative values for $n^{\text{✗}}$ and $n^{\text{✓}}$ for a range of reasonable (and fairly low) values for j, k and d and using $\ell = 1$ (ignore for now the bottom part and the column labelled $n(m)$).

2.4 Further Analyses

The full journal paper [Winikoff and Cranefield, 2014] also (a) develops a recurrence relation formulation that allows us to understand how the number of paths is affected by the number of available plans (j); (b) considers the *probability* of failing, and shows, unsurprisingly, that failure handling reduces the probability of a failure at some point in the goal-plan tree resulting in the whole execution failing; (c) examines how bounding the *rate* of failure (the number of action failures divided by the length of the path) affects the number of paths; the results show that, due to failure handling, most paths have a failure rate greater than 0.4, but the number of paths with lower failure rates is still extremely large; and (d) extends the analysis to deal with recursive trees of arbitrary (non-uniform) shape, given a specified bound on the length of traces, finding that there is a similar number of paths in a simple recursive tree as there is in uniform trees with the same path length.

2.5 Comparison with Procedural Programs

In order to argue that BDI programs are *harder* to test than non-agent programs we need to analyze the number of paths in non-agent programs, and compare with those in agent programs. We define a simple (abstract) procedural program as consisting of primitive statements, sequences of programs, or selection between programs, and then define the number of paths in a program P as $n(P)$. The key question then is: does a procedural program with m nodes have significantly fewer paths than a BDI program of the same size[7]?

We therefore define $n(m)$ as being the largest number of paths possible for a program of size m (formally: $n(m) \equiv \max\{n(P) \ : \ |P| = m\}$). The full paper derives a definition for $n(m)$. Table 1 (right-most column) shows values for $n(m)$ where m is the number of actions in the goal-plan tree.

[7] We define the size of a procedural program, denoted $|P|$, as being the number of primitive statements, and, comparably, the size of a BDI program as the number of actions.

Parameters			Number of			No Failure Handling		With Failure Handling		
j	k	d	goals	plans	actions	$n^{\checkmark}(g)$	$n^{\times}(g)$	$n^{\checkmark}(g)$	$n^{\times}(g)$	$n(m)$
2	2	3	21	42	62 (13)	128	614	6.33×10^{12}	1.82×10^{13}	6,973,568,802
3	3	3	91	273	363 (25)	1,594,323	6,337,425	1.02×10^{107}	2.56×10^{107}	5.39×10^{57}
2	3	4	259	518	776 (79)	1,099,511,627,776	6,523,509,472,174	1.82×10^{157}	7.23×10^{157}	2.5×10^{123}
3	4	3	157	471	627 (41)	10,460,353,203	41,754,963,603	3.13×10^{184}	7.82×10^{184}	5.23×10^{99}
Workflow with 57 goals(*)						294,912	3,250,604	2.98×10^{20}	9.69×10^{20}	$(\ell = 4)$
(*) The paper says 60 goals, but their						294,912	1,625,302	6.28×10^{15}	8.96×10^{15}	$(\ell = 2)$
figure 6 actually has 57 goals.						294,912	812,651	9.66×10^{11}	6.27×10^{11}	$(\ell = 1)$

Table 1: Illustrative values for $n^{\checkmark}(g)$ and $n^{\times}(g)$ both without and with failure handling, and for $n(m)$. The first number under "actions" (e.g. 62) is the number of actions in the tree, the second (e.g. 13) is the number of actions in a single execution where no failures occur. The bottom section shows numbers for the goal-plan tree in Figure 6 of Burmeister *et al.* [2008].

It is worth emphasising that $n(m)$ is defined as the *maximum* over *all* possible programs of size m. However, the maximal program is highly atypical. For example, considering all programs with seven statements, there are a total of 8,448 possible programs, but only 32 of these have 12 paths (the maximum). Indeed, the mean number of paths for a seven statement program is 4.379, and the median is 4. Overall, looking at Table 1, we conclude that the number of paths for BDI programs is much larger than even the (atypical) maximal number of paths for a procedural program of the same size. This supports the conclusion that BDI programs are *harder* to test than procedural programs.

3 A Reality Check

In the previous section we analysed an *abstract* model of BDI execution. But in relating this analysis to real systems there are two questions to be considered. Firstly, is the analysis *faithful* to the semantics of real BDI platforms (i.e. it does not omit significant features, or contain errors)? We checked faithfulness by comparing our abstract BDI execution model with results from a real BDI platform, namely JACK [Busetta *et al.*, 1999]. This comparison was done by encoding two example goal-plan trees in JACK, using a harness to generate all possible executions. The JACK code and our model produced exactly the same traces.

The second question is to what extent the large numbers in Table 1 apply to real applications? We investigated this by considering a goal-plan tree from a real industrial application, specifically the goal-plan tree (Figure 6) of Burmeister *et al.* [2008], which has "*60 achieve goals in up to 7 levels. 10 maintain goals, 85 plans and about 100 context variables*" (page 41). The bottom part of Table 1 gives the various n values for this goal-plan tree, for $\ell = 4$ (top row), $\ell = 2$ (middle row) and $\ell = 1$ (bottom row). With 57 goals, the tree has size in between the first two rows of Table 1. Comparing the number of possible paths in the uniform goal-plan trees against the real (and non-uniform) goal-plan tree, we see that the number is somewhat smaller in the real tree, but that it is still quite large, especially in the case with failure handling. However, we do note that their goal-plan tree only has plans at the leaves, which reduces its complexity: a goal-plan tree that was more typical in having plans alternating with goals would have a larger number of possible paths. Furthermore, the numbers for their goal-plan tree are a conservative estimate, since we assume that leaf plans have only simple behaviour, whereas it is clear that their plans are in fact more complicated, and can contain nested decision making (e.g., see their Figure 4). In other words, the number of paths calculated is an under-estimate of the actual number of paths in the real application.

4 Conclusion

Our analysis found that the number of possible paths for BDI agents is, indeed, large, both in an absolute sense, and in a relative sense (compared with procedural programs of the same size). As expected, the number of possible paths grows as the tree's depth (d) and breadth (j and k) grow. However, somewhat surprisingly, the introduction of failure handling makes a very significant difference to the number of paths. Before we consider the negative consequences of our analysis, it is worth highlighting one positive consequence: our analysis provides quantitative support for the long-held belief that BDI agents allow for the definition of highly flexible and robust agents.

So what does the analysis in this paper tell us about the testability of BDI agent systems? Consider testing of a whole system. The numbers depicted in Table 1 suggest quite strongly that attempting to obtain assurance of a system's correctness by testing the system as a whole is not feasible. Furthermore, the space of *unsuccessful* executions is particularly hard to test, since there are many unsuccessful executions (more than successful ones), and the probability of an unsuccessful execution is low, making this part of the behaviour space hard to "reach". What about unit testing and integration testing? Unfortunately, it is not always clear how to apply them usefully to agent systems where the interesting behaviour is complex and possibly emergent. A key consequence of emergence is that "more is different" which can make unit testing less useful.

Overall, we are in the position where there is further work to be done (e.g. making testing more sophisticated, and improving formal methods [Dastani *et al.*, 2010]), but currently we have no choice but to proceed with caution. That is, to accept that BDI agent systems are in general robust, but that there is, at present, no practical way of assuring that they will behave appropriately in all possible situations.

References

[Benfield *et al.*, 2006] S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In P. Stone and G. Weiss, editors, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 10–15. ACM Press, 2006.

[Bratman *et al.*, 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[Bratman, 1987] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[Burmeister *et al.*, 2008] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-agents for agile goal-oriented business processes. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS) [Industry Track]*, pages 37–44. IFAAMAS, 2008.

[Busetta *et al.*, 1999] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents — Components for Intelligent Agents in Java. AgentLink News (2), 1999.

[Dastani *et al.*, 2010] M. Dastani, K. V. Hindriks, and J-J. Ch. Meyer, editors. *Specification and Verification of Multi-agent systems*. Springer, Berlin/Heidelberg, 2010.

[de Silva and Padgham, 2004] L. P. de Silva and L. Padgham. A comparison of BDI based real-time reasoning and HTN based planning. In G.I. Webb and X. Yu, editors, *AI 2004: Advances in Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 1167–1173. Springer, Berlin/Heidelberg, 2004.

[Ekinci *et al.*, 2009] E. E. Ekinci, A. M. Tiryaki, Ö. Çetin, and O. Dikenelli. Goal-oriented agent testing revisited. In M. Luck and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 173–186, Berlin/Heidelberg, 2009. Springer.

[Erol *et al.*, 1994] K. Erol, J. A. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, pages 1123–1128. AAAI Press, 1994.

[Erol *et al.*, 1996] K. Erol, J. A. Hendler, and D. S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Gomez-Sanz *et al.*, 2009] J. J. Gomez-Sanz, J. Botía, E. Serrano, and J. Pavón. Testing and debugging of MAS interactions with INGENIAS. In M. Luck and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 199–212, Berlin/Heidelberg, 2009. Springer.

[Mathur, 2008] A. P. Mathur. *Foundations of Software Testing*. Pearson, 2008. ISBN 978-81-317-1660-1.

[Müller and Fischer, 2014] J. Müller and K. Fischer. Application impact of multi-agent systems and technologies: A survey. In O. Shehory and A. Sturm, editors, *Agent-Oriented Software Engineering*, pages 27–53. Springer Berlin Heidelberg, 2014.

[Munroe *et al.*, 2006] S. Munroe, T. Miller, R.A. Belecheanu, M. Pechoucek, P. McBurney, and M. Luck. Crossing the agent technology chasm: Experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, 21(4):345–392, 2006.

[Nguyen *et al.*, 2009] C. D. Nguyen, A. Perini, and P. Tonella. Experimental evaluation of ontology-based test generation for multi-agent systems. In M. Luck and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 187–198, Berlin/Heidelberg, 2009. Springer.

[Padgham and Winikoff, 2004] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004. ISBN 0-470-86120-7.

[Padgham *et al.*, 2013] L. Padgham, Z. Zhang, J. Thangarajah, and T. Miller. Model-based test oracle generation for automated unit testing of agent systems. *IEEE Transactions on Software Engineering*, 39:1230–1244, 2013.

[Rao and Georgeff, 1991] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, pages 473–484. Morgan Kaufmann, 1991.

[Sardina and Padgham, 2011] S. Sardina and L. Padgham. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23:18–70, 2011.

[Winikoff and Cranefield, 2014] M. Winikoff and S. Cranefield. On the testability of BDI agent systems. *Journal of Artificial Intelligence Research (JAIR)*, 51:71–131, 2014.

[Wooldridge, 2002] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester, England, 2002. ISBN 0 47149691X.

[Zhang *et al.*, 2009] Z. Zhang, J. Thangarajah, and L. Padgham. Model based testing for agent systems. In J. Filipe, B. Shishkov, M. Helfert, and L. Maciaszek, editors, *Software and Data Technologies*, volume 22 of *Communications in Computer and Information Science*, pages 399–413, Berlin/Heidelberg, 2009. Springer.

[Zhu *et al.*, 1997] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, December 1997.