Evolution of Agent-Oriented Distributed Model for Software Testing: A Layered Approach

Dhavachelvan Ponnurangam[1] and Uma Anbarasan[2]
[1] Department of Information Technology, Sri Manakula Vinayagar Engineering College, India
[2]Department of Computer Science and Engineering, Anna University, India

Abstract: *As new requirements arise, on one hand, from the increasing complexity of modern software systems and, on the other hand, from the distribution of today's information economies, it has been recognized that the modularity and reusability provided by existing techniques and approaches are insufficient. Although, each paradigm has its own contribution in the software engineering field on the support of their proficiencies, due to the exceptional growth of the software industry, researchers continue to strive for more efficient and powerful techniques. Agents are being advocated as a next generation model for engineering complex and distributed systems. They facilitate the automated software testing by virtue of their high-level decomposition, independency and parallel activation. Here, we address a set of more specific characteristics of agent-based approach (modularity, independency and parallel activation) and its efficacy in software testing. In this paper, we did not only describe the claims for agent-based approach in software testing, but also developed a multi-agent system for software testing with agent qualities. The multi-agent system illustrated here is on the basis of few basic operational real-world testing techniques, as an attempt to describe how to practice Agent-Oriented Software Testing (AOST) which has not previously done.*

Keywords: *Software testing agent, distributed testing framework, AOST, multi-agents.*

Received October 4, 2004; accepted April 30, 2005

1. Introduction

Delivering high quality software for real-world applications is difficult. A wide range of software engineering paradigms have been recently devised (e. g., object-orientation [5, 25], component ware [20], design patterns [3, 10] and software architectures [3, 11]) either to make the engineering process easier or to extend the complexity of applications that can feasibly be built [19]. As new requirements arise, on the one hand, from the increasing complexity of modern software systems, and on the other hand, from the distribution of today's information economies, it has been recognized that the modularity and reusability provided by other techniques and approaches are insufficient. Although each paradigm has its own contribution in the software engineering field on the support of their proficiencies, due to the exceptional growth of the software industry, researchers continue to strive for more efficient and powerful techniques [5, 19, 20]. Agents are being advocated as a next generation model for engineering complex and distributed systems [14, 19, 20, 21].

Several approaches to agent-oriented software engineering have been developed, ranging from structured, informal methodologies, to formal ones [3]. The explanations of Agent-Oriented Software Engineering (AOSE) [5, 7, 13, 14, 19, 20] are lacking in details that would allow a software tester to decide easily how to ship to agent-based software testing. Due to the above claim, there has been comparatively little work on agent-based testing as a serious software engineering paradigm that can significantly enhance development in wide range of applications. These shortcomings can be rectified by recasting the essential components of agent systems into more traditional software engineering concepts [4, 7, 19]. Here we address a set of more specific characteristics of agent-based approach (modularity, independency and parallel activation) and its efficacy in software testing.

Effective test automation can be achieved by dividing the testing components to a maximum possible limit and maintaining by different units with higher degree of independency [8, 12, 15, 29]. Agent technologies facilitate the automated software testing by virtue of their high-level decomposition, independency and parallel activation. In this paper, we did not only describe the agent-based approach in software testing, but also developed a multi-agent system for software testing with agent qualities. The multi-agent system illustrated here is on the basis of few basic operational real-world testing techniques, as an attempt to describe how to practice agent-based software testing, which has not previously done. The advantages of multi-agent systems are that they can compartmentalize specialized task knowledge, organize them to avoid processing bottlenecks, and can be built expressly to deal with dynamic changes in the agent environment [6].

Defining and classifying a relatively new phenomenon is always a difficult task to face the objections of basic definitions, arguments that important points have been overlooked, or claims that are not really new anyway [1, 7, 16, 19, 22, 23]. Bringing together agents and other fields of software engineeering might be difficult as the advantages of agent technology are still not widely recognized. The method discussed here is to offer a definition for encompassing to cover the software testing phenomena, based on agents, at the preliminary level, yet sufficiently tight that it can rule out complex systems that are clearly not agent-based. This paper therefore provides a timely summary and enhancement of agent theory in software testing, which motivates recent efforts in adapting concepts and methodologies for Agent-Oriented Software Testing (AOST) to complex systems.

This paper is organized as follows. Section 2 describes the Multi-Agent System (MAS) for software testing. In section 3, experimental results are discussed. Finally, in section 4, conclusion and future perspectives are presented.

2. Construction of Multi-Agent System

2.1. Background Information Needed

Definition 1: Let S be the MAS constructed for providing variety of testing environments and it can be defined as, $S = \{D_1, D_2, ..., D_Z, S_1, S_2, ..., S_X..\}$, where D is the distributor agent, S is the testing agent and X is the number of testing agents and also the number of testing techniques available in the system. Let A be the set of agents needed for the product P and it can be defined as, $A\{D_P, a_1, a_2, ..., a_y\}$ where y is the number of testing agents and also the number of testing techniques needed by the product P.

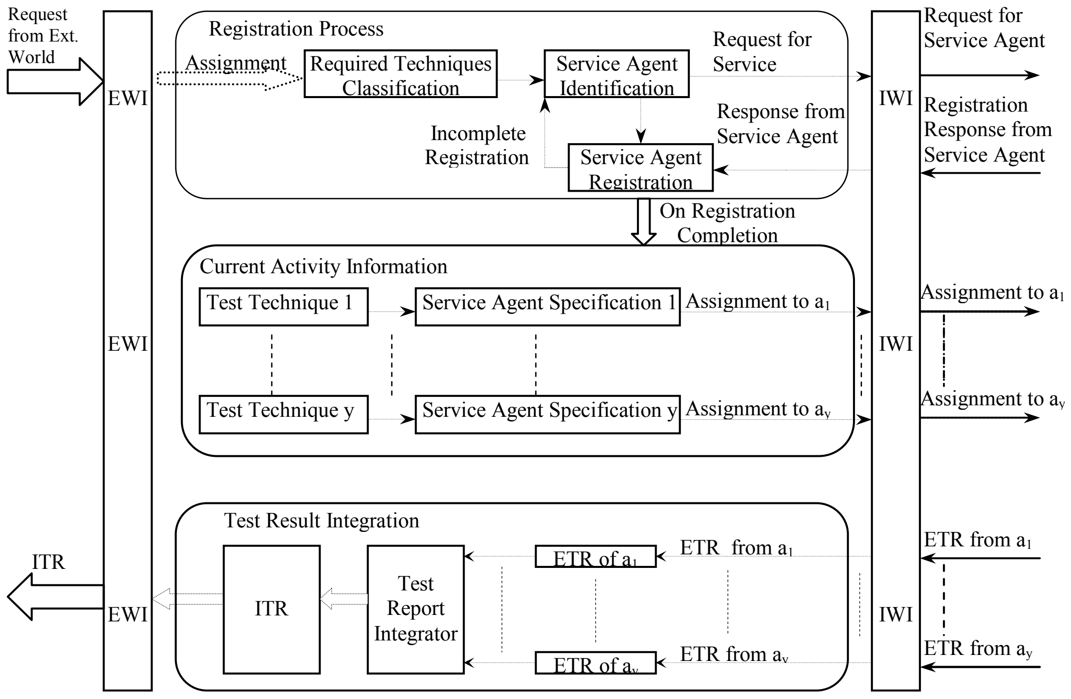
If there are multiple products to be tested simultaneously, then P and A can be extended as $\{A_1, A_2, ... A_H\}$ and $\{P_1, P_2, ... P_H\}$ respectively, where, H is the number of products to be tested simultaneously. In such cases, $A_q \cap A_r$, where, $0 < q, r \leq H$ and $q \neq r$. i. e., at any specific service duration, there is no single agent (distributor) or agent set (testing agent + clones) that can be shared by more thanone product simultaneously. This improves the autonomous property and fault tolerance of the agents.

Sometimes the input to the distributor agent may also consist of time specification. i. e., T_P is the permitted time to complete the testing processes. In the testing agents in A, there will be the predicted values about the number of test cases to be formed and executed, and the average time for single test case execution (based on the program attributes such as data variables, statements, functions, independent paths, etc). Let C_j be the total number of test cases for a_j, $0 < j \leq y$. Let t_{g_j} be the average test case generation time, t_{e_j} be the average test case execution in a_j and T_j is the total time to be spent in a_j.

2.2. Multi-Agent System for Software Testing

2.2.1. Defining Agents' Behavior

- *Distributor Agent:* Distributor agent D depends on the testing agents and their clones for the testing to be performed, or the testing resource to be made available. D can get the input as the combination of P, specification about A and an optional piece as T_p. Based on the specification about A, D can select the components of A from S and distribute the service based assignment to all of them. This decomposition and distributions allows one to apply alternative coordination mechanisms (such as cloning by testing agents, direct supervision of testing agents over their corresponding clones) and generating the decision parameters (such as C_j, K_j for clones generation and load scheduling) in order to achieve a literal MAS. The message from D to a_j is a set of {P, T_p, specification about a_j}. The response from a_j to D, will consist of {environmental integrated reports (number of test cases, number of faults detected), specification about a_j (type of testing technique)}. The output of D consists of {specification about A (such as types of testing techniques) + environmental integrated reports (number of test cases for each technique, number of faults detected, techniques based performance) + integrated test reports (total number of test cases, effort spent, total number of faults detected)}. The primary components and the information flow within and around the distributor agent are illustrated in the Figure 1.
- *Testing Agent:* The testing agents depend on the distributor agent to get the assignment with specifications. Also, they depend on their clones for the task to be performed within a precise time period that depends on the type of the task. Clones generation and load sharing are based on the time specification T_p supplied by D. So, the agents must estimate how long it would take them to complete the job that is assigned to them. For this, we can build a 'predictor' to make these estimates. There is still the issue of how the originating agent makes an estimate of the jobs completion time. The predictor can estimate number of different paths to be tested (basis path testing), number of data variables to be assessed (data flow testing), number of loops to be exercised (loop testing), number of conditions to be verified (control flow testing) [26]. Based on the above factors, the predictor can predict the number of test cases to be generated and executed and the time duration required for it [17, 18]. The components and working principle of testing agent is explained in Figure 2.


Request from Ext. World
EWI
Registration Process
Assignment
Required Techniques Classification
Service Agent Identification
Request for Service
Response from Service Agent
Incomplete Registration
Service Agent Registration
IWI
Request for Service Agent
Registration Response from Service Agent
On Registration Completion
Current Activity Information
Test Technique 1
Service Agent Specification 1
Assignment to a1
Test Technique y
Service Agent Specification y
Assignment to ay
Test Result Integration
ITR
Test Report Integrator
ETR of a1
ETR of ay
ETR from a1
ETR from ay

EWI: External World Interface, i. e., interface with the users.
IWI: Internal World Interface, i. e., interface with other system components.
ITR: Integrated Test Report, i. e., consolidated output from the distributor.
ETR: Environmental Test Report, i. e., technique based output from the Individual testing agent.
——: Information flow inside the agent.
……: Information outside the agent.

Figure 1. Information flow related with distributor agent.

a. *Job Origination:* The task assignments for the testing agents are to be delivered by the distributor D. After receiving the assignments from D, each agent can manage and control itself on a local dimension and interact directly with its clones to exchange, provide and receive services, data and knowledge. But the testing agents need not to communicate with each other. Similarly there is no interaction to be maintained between the clones of testing agents. The details are hidden to each other. Testing agents must have prediction modules to compute about C_j, K_j for test scheduling. The testing agents have independent options to be operated either in the testing mode or in the distributor mode and that should be based on the time parameter supplied by the distributor agent and the structural properties of the software being tested.
b. *Testing Mode:* In this mode, the testing agents will be performed to execute the test cases and at the end, send the technique specific integrated test report to D. Here a_j needs to generate clones and A must be the subset of S. i. e., $A \in S$ and $y \le x$. This is explained in the step-3.2. in the cloning algorithm. In this mode, the total time to be spent in a_j can be calculated as in equation (1) as follows.

$$T_j = C_j * (t_{g_j} + t_{e_j}) \tag{1}$$

c. *Distributor Mode:* The clones of any agent must be controlled by their respective parent agents. i. e., all testing agents (except clones) must be capable enough to act as a distributor and as a load scheduler for their respective clones. The message from a_j to its clones might be a set of {P or part of P + specification about ac_j}. After the completion of assigned task to the clones, the response transmission to the parent agent will consist of {Reference to Partial P + specification about ac_j + Individual test Results}. Then the parent agent will collect the individual test results from its clones and will generate the integrated test report for any particular testing environment. This mode is explained in the step-3.3 in the cloning algorithm. In this mode, the testing agent a_j can be defined as, $a_j = \{a_j, ac_{j1}, ac_{j2}, ...ac_{jK_j-1}\}$ where K_j indicates the total number of agents in the specific testing environment. K_j-1 denotes the number of cloning agents for testing purpose only and the remaining one a_j (testing or parent agent) will act as environmental distributor (distributor only for specific testing technique) not as D. If any agent aj is overloaded, the load must be shared by multiple identical agents. i. e., a_j must be cloned as ac_1, ac_2, and so on. Here, A needs not to be the subset of S. i. e., $A \notin S$ but $y \le x$. The total time to be spent for testing can be calculated by using equation (2) as follows.

$$T_j = (C_j / K_j) * (t_{g_j} + t_{e_j}) \tag{2}$$

2.2.2. Cloning of Agents

All the clones of a particular agent are identical in that they have exactly same behavior. Each principle partner (clone) can manage and control itself on a local dimension and interact directly with its originator to exchange, provide and receive services, data and knowledge. The structure of the agent is the reduced version of testing agent, which only consists of the 'testing section' and 'test result integrator' as in the testing agent.

Effect of Cloning: The addition of testing agents improves the adequacy criteria and enhances the defect detection rate with corresponding increment in the number of test cases. The addition of clones for particular environment will reduce the total processing time but there will be an increment in the number of testers in the manual testing. Of course, if one adds more agents (clones) in a particular environment, then the system throughput will always improve irrespective of the type of testing – either manual or automated. The cloning process can be done based on either T_p or units (classes) components, packages, etc. For the remainder of the simulation runs in this paper, the cloning process is based on the number of units (classes).

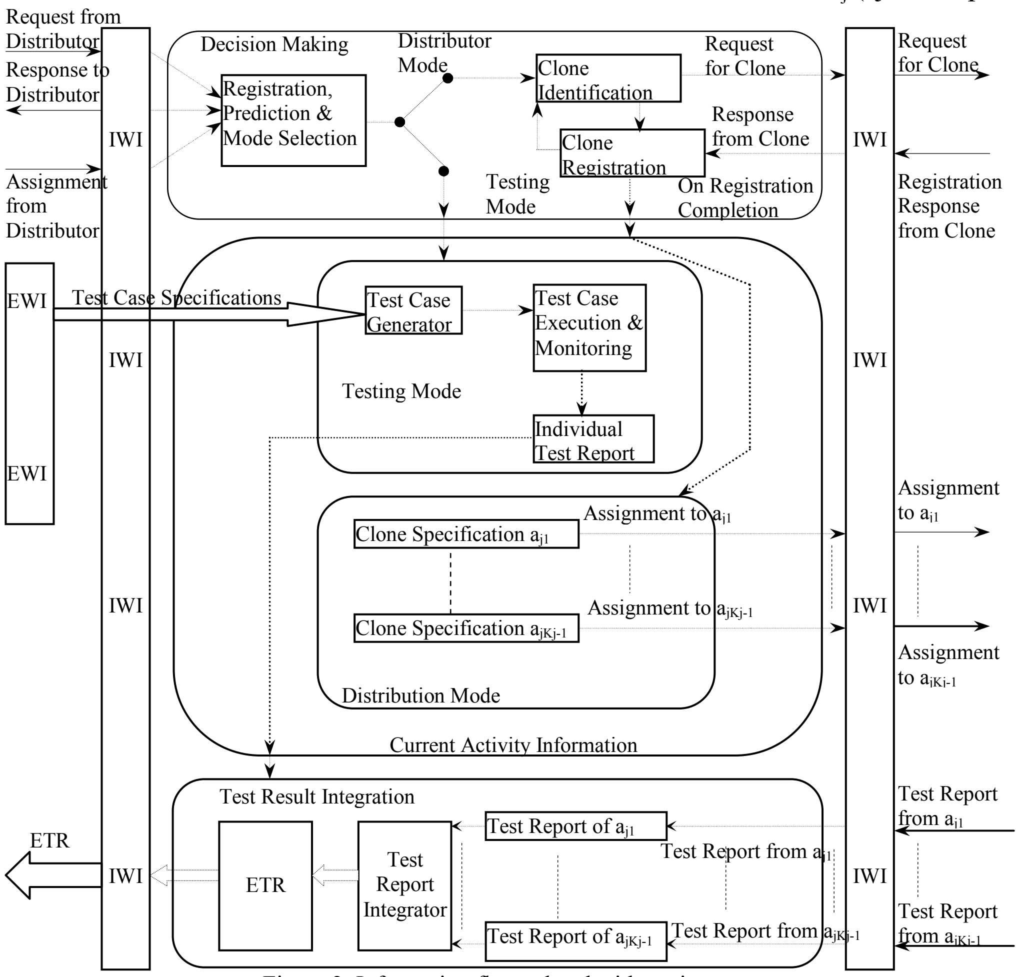
2.2.3. Request Negotiation Algorithm

Definition 2: With respect to the product P_q which is to be tested, the required set of agents can be defined as:

$$A_{P_q} = \left\{ \begin{array}{l} A_u \,|\, A_u = (D_u, a_{u1}, a_{u2}, \ldots\ldots. \, a_{uy}), \\ a_{uj} = (a_{uj}, ac_{uj1}, ac_{uj2}, \ldots\ldots. \, ac_{ujK_{uj}-1}), \\ 0 < q \leq F, 0 < u \leq H, 0 < j \leq y. \end{array} \right\} \quad (3)$$

Where the following hold:

- P_q: Is the product to be tested and F is the total number of requests waiting in the request queue for service. On the arrival of new requests, the request queue will be updated automatically and the count F also will be updated as F = F + 1.
- H: Is the number of products to be tested simultaneously.
- For successful service, i. e., agents allocation for the product P_q, A_{P_q} must be less than or equal to I, where I is the set of agents that are idle. i. e., not participated in the current service.
- a_{uj}: Is the one of the testing agent of the product u in H and $ac_{uj\ldots}$ is the one of the clone of a_{uj}.
- K_{uj}: Is the total number of agents in the particular testing environment of u and K_{uj}-1 denotes the number of clones of a_{uj} (aj of the product u).


Request from Distributor
Response to Distributor
Assignment from Distributor
IWI
Decision Making
Registration, Prediction & Mode Selection
Distributor Mode
Clone Identification
Clone Registration
Request for Clone
Response from Clone
Testing Mode
On Registration Completion
Request for Clone
Registration Response from Clone
EWI
Test Case Specifications
Test Case Generator
Test Case Execution & Monitoring
Testing Mode
Individual Test Report
EWI
Clone Specification a_{j1}
Assignment to a_{j1}
Clone Specification a_{jKj-1}
Assignment to a_{jKj-1}
Distribution Mode
Current Activity Information
Assignment to a_{i1}
Assignment to a_{iKi-1}
Test Result Integration
ETR
Test Report Integrator
Test Report of a_{j1}
Test Report from a_{j1}
Test Report of a_{jKj-1}
Test Report from a_{jKj-1}
Test Report from a_{i1}
Test Report from a_{iKi-1}
Figure 2. Information flow related with testing agent

Request negotiation algorithm is explained as follows:

```
1. Initialization
   1.1. $I = S$, $H = 0$, $q = 0$.
   1.2. Receive the new requests and set F with
        appropriate value.
2. 2.1. $q = q + 1$.
   2.2. Get Request ($P_q$).
   2.3. Define $A_{P_q}$ as in (3).
   2.4. If $A_{P_q} \leq I$,
        2.4.1. Allocate agents for the product $P_q$
               as defined in the step-2.3.
        2.4.2. Update I as $I = I - A_{P_q}$.
        2.4.3. $H = H + 1$.
        2.4.4. Update F as
                 $F = F - Request (P_q)$
   2.5. If $A_{P_q} > I$, Request ($P_q$) cannot be
          processed temporarily. Goto step-3.
3. If $q > F$, $q = 0$.
4. If $F \neq 0$, then goto the step-2.1.
5. Stop the process.
```

3. Experimentation and Analysis

The minimal version of the proposed framework is constructed and tested in a LAN set up in the concept proving stage [3]. The test samples are implemented in C++ as experimental projects by two different teams. Team 1 consists of four students and headed by an academician and the team 2 consists of four students and headed by a software developer. For each version of the same project, the number of test cases is defined as a fixed package for a particular testing technique.

As said earlier, the framework is constructed on the basis of few basic operational real-world testing techniques, as an attempt to describe how to practice agent-oriented software testing. The experiment was carried out in different environments (testing techniques) namely loop testing (testing loops only), condition testing (testing conditions only) and data flow testing (testing data variables only).

The significance of the proposed framework can be realized by analyzing the experimental values of total number of test cases for each technique (C_j), time spent for particular testing technique (T_j), errors found (E_j) by using C_j, and the number of agents/persons involved in particular testing K_j from the Table 1. In this minimal version, the clone generation is based on the number of classes (one class-one agent) and not based on T_p.

From the estimates and observed values, the primary advantages of the agent-decomposition, independency and parallel activation are realized. From the estimates of C_j, T_j, E_j and K_j-1 under different situation, it is observed that the reliability of the software can be enhanced by applying the framework and considerable amount of time can be saved which will be needed as the software approaches shipping without any degradation with respect to the reliability of the software irrespective of number of technical persons involved.

Table 1. Test samples description and test reports.

Test Statistics			Type of Projects							
			Game		Editor		Laboratory Experimental Pack		Medical Image Analysis Tool	
			Team 1	Team 2	Team 1	Team 2	Team 1	Team 2	Team 1	Team 2
Size in LOC			400+	500+	700+	700+	1600+	1800+	2500 +	2500+
Type of Testing (Loop Testing)	Without MAS	C_j	45	55	90	95	170	185	240	265
		T_j in hrs	5.5	8.0	13.0	14.0	25.0	28.0	36.0	40.0
		E_j	18	25	27	25	31	33	31	36
		K_j	1	1	1	1	1	1	1	1
	With MAS	C_j	45	55	90	95	170	115	240	265
		T_j in hrs	1.5	2.0	2.25	2.25	2.0	2.25	2.25	2.5
		E_j	25	26	28	26	36	39	37	42
		K_j	3	3	4	4	10	10	12	12
Type of Testing (Condition Testing)	Without MAS	C_j	69	84	114	123	294	336	414	438
		T_j in hrs	8.5	10	15	17	45	50	66	78
		E_j	18	23	38	36	58	64	24	26
		K_j	1	1	1	1	1	1	1	1
	With MAS	C_j	69	84	114	123	294	336	414	438
		T_j in hrs	2.25	2.5	3.0	3.25	3.5	4.25	4.5	5.5
		E_j	21	25	45	48	69	76	31	37
		K_j	3	3	4	4	10	10	12	12
Type of Testing (Data Flow Testing)	Without MAS	C_j	28	34	48	51	82	89	117	121
		T_j in hrs	4	5.25	6.5	7.0	10.0	12.0	15.0	16.0
		E_j	18	24	19	26	29	35	42	46
		K_j	1	1	1	1	1	1	1	1
	With MAS	C_j	28	34	48	51	82	89	117	121
		T_j in hrs	1.0	1.25	1.25	1.5	1.0	1.25	1.0	1.0
		E_j	19	29	23	27	34	37	52	53
		K_j	3	3	4	4	10	10	12	12

4. Conclusion

The multi-agent system presented here is systematic and it does illustrate its effectiveness in selecting the appropriate assignment based on requirements. This methodology rests on the idea of building a conceptual model that is incrementally refined and it can be extended from other existing models of other fields of software engineering. The arguments and results support that the agent models fit better for testing the complex software systems. This allows the system to perform better than the existing non-agent systems in the face of high throughput. The interpretations offered here concentrate on necessary, rather than sufficient conditions, so that they can be extended. Other related work includes developing distributed algorithms for reorganizing when goals are not being met by the agents in the systems.

References

[1] Avison D., Lau F., Myers M., and Nielsen P. A., "Action Research," *Communications of the ACM*, vol. 42, no.1, pp. 94-97, January 1999.

[2] Basali V. R., Selby R. W., "Comaparing the Effectiveness of Software Testing Strategies," *Technical Report*, University of Marryland, College park, USA, 1985.

[3] Booch G., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, 1994.

[4] Chavez A., Mouka A., and Maes P., "Challenger: A Multi-Agent System for Distributed Resource Allocation," *in Proceedings of the 1st International Conference on Autonomous Agents*, Marina Del Ray, California, USA, pp. 323-331, 1997.

[5] CianCarini P. and Wooldridge M. (Ed), *Agent-Oriented Software Engineering*, vol. 1957, LNCS, Springer-Verlag, 2001.

[6] Decker K., Pannu A., Sycara K., and Williamson M., "Designing Behavior for Information Agents," *Technical Report*, The Robotics Institute, Carnegie Mellon University, Panama, July 1996.

[7] Dhavachelvan P. and Uma G. V., "Multi-Agent Based Integrated Framework for Intra-Class Testing of Object-Oriented Software," *accepted in Journal on Applied Soft Computing*, Elsevier, pp. 205-222, 2004.

[8] Duran J. W., Ntafos S. C., "An Evaluation of Random Testing," *IEEE Transactions on Software Engineering,* vol. SE-10, no. 4, pp. 438-443, July 1984.

[9] Franklin S. and Graesser A., "Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agent," *in Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages,* Springer-Verlag, pp. 197-203, 1996.

[10] Gamma E., Helm R., Johnson R., and Vlissides J., *Design Patterns*, Addison-Wesley, 1995.

[11] Henry S. and Goff R., "Complexity Measurement of a Graphical Programming Language," *Software Practice and Experience*, vol. 19, no. 11, pp. 1065-1088, 1988.

[12] Hetzel, William C., *The Complete Guide to Software Testing*, QED Information Sciences, 1998.

[13] Jennings N. R. and Wooldridge M. (Eds), *Agent Technology: Foundations, Applications and Markers*, Springer, Berlin, 1998.

[14] Jennings N. R., Wooldridge M., "Agent-Oriented Software Engineering," in Bradshaw J. (Ed), *Handbook of Agent Technology*, AAAI/MIT Press, 2000.

[15] Jones B. F., Sthamer H. H., and Eyres D. E. "Automatic Structural Testing Using Genetic Algorithms," *Software Engineering Journal,* vol. 11, no. 5, pp. 299-306, 1996.

[16] Kitchenham B. A., Pfleeger S. L., Hoaglin D. C., and Rosemberg J., "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, August 2002.

[17] Malaiya Y. K., Karunanithi N., and Verma P., "Predictability of Software Reliability Models", *IEEE Transaction on Reliability*, vol. 41, no. 4, pp.539-546, December 1992.

[18] Malaiya Y. K., Mayrhauser A. V., and Srimani P. K., "An Examination of Fault Exposure Ratio," *IEEE Transaction on Software Engineering*, vol. 19, no. 11, pp. 1087-1094, November 1993 .

[19] Nicholas R. Jennings, "On Agent-Based Software Engineering," *International Journal on Artificial Intelligence*, vol. 117, pp. 277-296, 2000.

[20] Perini A., Brescian P., Giorgini, P. Giunchigila F., and Mylopoulas J., "Towards an Agent-Oriented Approach to Software Engineering," *in Proceedings of Dagli Oggetti Agli Agenti* (*Woa)*, Modena, Italy, pp. 272-284, September 2001.

[21] Petrie C., "Agent-Based Software Engineering," *Lecture Notes in Artificial Intelligence*, Springer-Verlag, vol. 1957, pp. 58-76, 2001.

[22] Seaman C. B., "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, August 1999.

[23] Sjoberg D. I. K., Anda B., Arisholm E., Dyba T., Jørgensen M., Karahasanovic A., Koren E. F., and Vokac M., "Conducting Realistice Experiments in Software Engineeing," *in Proceedings of the International Symposium on Empirical Software Engineering (ISESE'02)*, IEEE Computer Society, pp. 17-26, 2002.

[24] Szyperski C., *Component Software*, Addison Wesley, 1998.

[25] Tracey N., Clark. J., and Mander K., "The Way Forward for Unifying Dynamic Test-Case Generation: The Optimization-Based Approach," *in Proceedings of the IFIP International Workshop on Dependable Computing and it's Applications (DCIA)*, South Africa, pp. 169-180, January 1998.

[26] Trivedi K. S., *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice Hall, India, 2003.

[27] Wagnor K., Chin C., and McCluskey E., "Pseudo Random Testing," *IEEE Transaction on Computers*, vol. C-36, pp. 332-343, March 1987.

[28] Wooldridge M. and Jennings N. R., "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10, no. 2, 1995.

[29] Zhu H., Patrick A. V. Hall, and John H. R., "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol.29, no. 4, pp. 367-427, December 1997.

Dhavachelvan Ponnurangam received his BE degree in electrical and electronics engineering from the University of Madras, India, in 1997 and his ME degree in computer science and engineering from the Anna University, India, in 2000. Currently, he is an assistant professor in the Department of Information Technology at Sri Manakula Vinayagar Engineering College, Pondicherry, India, and he has completed his PhD in the area of agent-based software testing, from the Department of Computer Science and Engineering, Anna University, India. His research interests include software testing, software metrics, and soft computing and related applications.

Uma Anbarasan received her ME and PhD degrees in computer science, both from the Anna University, Tamil Nadu, India, in 1993 and 2001, respectively. Currently, she is an assistant professor in the Department of Computer Science and Engineering at the Anna University of Tamil Nadu, India. She has published approximately 40 journal and congress papers, and she is the author of two books in computer science. Her research interests include neural networks, fuzzy systems, machine learning, and related applications. Her current main research interests are in the fields of fuzzy and linguistic modeling, knowledge based systems, and fuzzy expert systems.