JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2008**

Vol. 7, No. 4, May-June 2008

A (TRUSTAD) Component Nomenclature

Sivamuni Kalaimagal and **Rengaramanujam Srinivasan**
B.S.A. Crescent Engineering College, Chennai,India

Component based engineering is gaining substantial interest in the market today. This is because software components offer us the advantage of lesser developmental costs and shorter life cycles. With the continuing rise in the demand for software component based products, the terms - component, in-house component, COTS and reusable components have become overloaded and rather conflicting over the past years. This paper attempts to clearly define the above terms. We also propose a seven dimension vector (T, R, U, S, T, A, D) that can be used to specify software components. The seven dimension vector is then used to provide a set of specifications that distinguish between in-house components, COTS components , reusable in-house and reusable COTS components. Since, our nomenclature is based on this seven dimensional TRUSTAD vector; we have chosen to name the classification as the TRUSTAD nomenclature.

1 INTRODUCTION

Component based software development is increasingly gaining acceptance in the market today. The terms component, in-house components, COTS, reusable components have become overloaded and rather conflicting over the past few years [2]. A search on any internet based search engine gives a wide variety of results. Many research papers use the term components without explicitly specifying what kind of component they are talking about. The term components is very broad based and the demarcation between a component, in-house component, COTS and reusable component is quite fuzzy [16]. Since, components, in-house components and COTS components mean different things to different people, first we give alternative definitions already in vogue. Then we offer our structured definitions for these terms, using the TRUSTAD specification proposed by us.

The paper is organized as follows: Section 2 provides a definition for a software component. Section 3 lists out the different types of software components. Section 4 provides the proposed TRUSTAD specification for a software component. Section 5 defines and explains the specification for an in-house component. Section 6 defines and explains the specification for a COTS component. In section 7, we have attempted to put forth our definition for reusable in-house and COTS components . Section 8 gives a consolidated view of the TRUSTAD nomenclature.

2 SOFTWARE COMPONENTS DEFINITION

Let us consider the different definitions put forth so far, for a software component. According to C.Szyyperski [9] , "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties." Not all software components can be subject to composition by third party vendors. Hence, the above definition though technically correct, is not complete. Michael Sparling [2] defines a component, "as a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container accessed via one or more published interfaces. While a component may have the ability to modify a database, it cannot be expected to maintain state information. A component is not platform constrained nor is it application bound." Only recently available technologies like JavaBeans and EJB deploy components in containers and it is not a given rule that all components exist in containers. So, we tend to disagree with this definition also. Before, we present our definition for a component; let us try to understand what a software component is.

Let us consider an analogy first. When we purchase a computer, we generally buy different parts from different vendors and assemble them together. We may get the hard disk from Seagate, monitor from Samsung, the keyboard from HCL and we can top it all with a P4 processor from INTEL. The general idea is that we get the components suited to our specifications and by assembling them all together, the final product is cost effective as compared to a wholly branded model, that may not snug fit to our design specifications. Even in the automobile industry, the different parts of a car are manufactured by different vendor factories, and then assembled together to get a final product, thus cutting down on production costs.

On the other hand, let us consider a software program for an Airline Reservation System with separate modules for ticket reservation, ticket cancellation and ticket availability. A decade ago, it would have been impossible to get each module developed by a different vendor and then assemble them together to get a fully integrated product [16]. This is because software can be written in different languages and different platforms and it was not possible for software written in one platform to be integrated with software written in another platform [20]. This brought about the component revolution.

Component technology allows software modules written in different technologies to be integrated with one another, with the help of middleware technologies like COM, DCOM, CORBA, JavaBeans and EJB. These middleware technologies are simply a set of specifications or rules in the form of functions, which when incorporated into the code allows the software to be integrated with software developed using other platforms/languages [1].

To explain further, if the ticket reservation module in an airline reservation system had been written using VB following the COM/DCOM specifications we could have integrated it with any other airline reservation application program written in any other language/platform that needed a ticket reservation module. This saves development time for the programmers and may also provide cost benefits for the organization [15] [20].

Therefore, there are a few things which should be taken into consideration, before framing a definition for a component. The first important point to remember is that, once a piece of software has been written, using such a middleware specification, it can be integrated only if it has already been executed and tested. So, we have a group of persons who develop software components (component developers) and a group of persons who use software components (component users).

The second point to remember is that the component user cannot access the software component directly, but only through a set of well published interfaces for security reasons [12]. Developers would not like every person to have access to their source code for obvious reasons. A component‘s interfaces are always independent of the implementations [13], [4].

The third point to remember is that components created using middleware technologies like JavaBeans and EJB are encapsulated inside a *container*[24],[25].

Consolidating the above three points, we present our definition of a component as: “Any piece of independently executable binary code written to a specification, which can only be accessed via a set of well published interfaces and which can be integrated into any kind of software application irrespective of language /platform. A component always offers a set of services via its interfaces and may be encapsulated inside a container depending on the kind of middleware technology used to develop the component.”

Now, that we have formally defined components, let us have a look at the different types of components.

3 SOFTWARE COMPONENT TYPES

Software components can be broadly classified into two categories: In-house components that can be developed inside the organization itself and COTS (commercial off the shelf) components that are purchased from third party vendors.

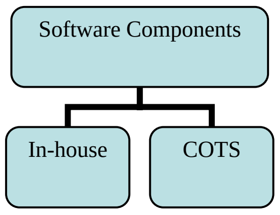

Figure 1: Software Component Types

4 SOFTWARE COMPONENT SPECIFICATION

The authors of [11] have put forth a proposal for specifying COTS. Based on that, we have put forth our specification for a software component, by making modifications to the features suggested by them and by adding three more features-testing type, accessibility level and reusability. We have used this specification to attempt to distinguish between in-house components, COTS components, reusable in-house components and reusable COTS components.

We suggest, that a software component can be specified using a 7 dimension vector (T, R, U, S, T, A, D).Each letter represents a component feature and each feature in turn has a set of attributes. The expansion and explanation of each category and attribute is shown in Table 1.

S.No	Feature	Attribute
1.	**T**esting Level	Test Type
2.	**R**eusability	Locatability
		Extensibility
3.	**U**sage	Functionality
4.	**S**ource	Origin
5.	**T**ype of Delivery	Packaging Type
6.	**A**ccessibility Level	Degree of Access
7.	**D**egree of Customization	Modification Degree
		Interface Documentation

Table 1- Component Specification

The different features and attributes can be explained as follows:

1) Testing Level

The testing level feature of a component describes how far the component can be tested. The attribute for this component is:

- **Testing Type**

  This describes what type of tests can be done. The values can either be Black box testing, White box testing or both.

2) Reusability

This feature describes how conducive the component is to being reusable. The attributes for this feature are:

- **Locatability**
  Locality describes the degree of ease with which the correct component can be made available from a component repository. The values for this are easy to locate, and difficult to locate.
- **Extensibility**
  Extensibility is whether the component can be extended to suit the need of the application it is going to be used with. The values for this are higher degree and lower degree.

3) Usage

The Usage feature is the mode in which the component is made use of. The attribute for this feature is:

- **Functionality**
  Functionality describes the scope of application of the component. The values for this can be domain specific or generic.

4) Source

The Source feature describes where the component comes from. The attribute for this feature is:

- **Origin**
  Origin describes where the product is developed. The possible values for this can either be internal (developed inside the organization) or external (developed outside the organization).

5) Type of Delivery

This component feature describes the form in which the component is delivered. The attribute for this feature is

- **Packaging Form**
  Packaging form describes how the component is packaged. The values are source code, statically linkable binary library, dynamically linkable binary library, binary component and stand alone executable program.

6) Level of Accessibility

This feature describes the depth to which the component can be accessed. The feature for this attribute is:

- **Degree of Access**
  This attribute describes how much of information is hidden to the component user .The values are lesser degree and greater degree of Information hiding.

7) Degree of Customization

The above component feature describes how flexible the component is with regard to customization. The two attributes for this are degree of modification and interface documentation.

- **Degree of Modification:**
  This is about the level of modification that can be done to the component. The values for this attribute are *Extensive Reworking*, *Internal Code Revision*, *Programming*, *Customization* and *Parameterization* where parameterization is on the lowest level of the scale and Extensive reworking is on the highest level.
- **Interface Documentation:**
  Interface documentation is whether the interface is provided with good documentation or not. The values can either be *mandatory* or *optional.*

We use the above specification for a software component, to formally present our definition of an in-house, COTS, reusable in-house and reusable COTS components based on the values they have for the attributes in the seven dimensional vector specified above.

5 IN-HOUSE COMPONENT SPECIFICATION

An In-house component can be defined as "*Any software component that has been developed for a particular application, either by the team that requires the components or any other alternate team, but within the same organization itself*".

This definition leads to the following conclusions namely that

1. Since in-house components, are developed within the organization itself, testing is easier because resource people will be easily available.
2. Also, the source code for in-house components can also be made available to the component users since they are all from the same organization itself.
3. Maintenance of the component will also not be much of a problem, because there is no fear of going to a third party vendor for maintenance. Any problem during component integration can be handled within the organization itself.
4. A full description of the component's behavior can be made available to the component user at any time.

Based on the above facts, the specification for an in-house component is given in table 2 below. As seen in the Table 2, In-House components are developed inside the organization itself. This means that the developers of the component are at a close proximity and hence the source code of the code can be made available if the situation arises.White box testing as well as black box testing can be done because access to the source code is possible. The degree of customization is greater. Maintenance and access to the source code is also possible because it will be easier to contact the component developers.

Category	Attribute	Value
Testing Level	Testing Type	1) Black Box Testing 2) White Box Testing
Reusability	Locatability	1) Easy to locate
	Extensibility	2) Higher degree of extensibility.
Usage	Functionality	Domain Specific
Source	Origin	Internal
Type of Delivery	Packaging Type	1) Source code 2)Statically linkable binary library 3) Dynamic linkable binary library 4)Binary Component 5) Stand Alone Executable Program
Accessibility Level	Degree of Access	1)Lesser degree of information hiding
Degree of Customization	Modification Degree	1) Extensive Reworking 2)Internal code Revision 3)Programming 4) Customization 5) Parameterization
	Interface Documentation	1) Optional

Table 2- In-House Component Specification

The definition and specification of Commercial-off-the-shelf (COTS) components is given in the next section.

6 COTS SPECIFICATION

COTS components are defined by Vigder and Dean as "*components which are bought from third party vendors and integrated into the system*" [4].However, according to [5] a more detailed and expanded view of COTS components should be taken. A

COTS component could be as small as a routine that computes the square root of a number or as large as an entire library of functions. The important thing is that a COTS component already exists and was created by people outside the software development organization that will actually use it [5].

A commercial – off-the-shelf component can therefore be defined as,” any software component that already exists, that was created by people outside the organization that will be using it, and that was purchased from a third party vendor.” A COTS component can be as small as a function to calculate the exponential of a number or it can be as large as credit card validation software.

The above definition leads to the following conclusions:

1. The source code can never be made available to the component user, unlike an in-house component. A COTS component is like a black box. The user can view the component only through its interfaces and outputs.
2. There are possibilities that a complete description of the component’s behavior may not be given by the vendor to the user. This may result in problems during component integration.
3. Maintenance can become an issue because the vendor may not correct defects or add enhancements according to the component’s specification.
4. Sometimes the vendors can provide updated components that do not integrate with the earlier version of the application. A classical example is the Ariane V that is usually quoted in component literature [7].

Category	Attribute	Value
Testing Level	Testing Type	1) Black Box Testing
Reusability	Locatability	1) Easy to locate
	Extensibility	2) Higher degree of extensibility
Usage	Functionality	Domain Specific
Source	Origin	External
Type of Delivery	Packaging Type	1)Statically linkable binary library 2) Dynamic linkable binary library 3)Binary Component 4) Stand Alone Executable Program
Accessibility Level	Degree of Access	1)Greater degree of information hiding
Degree of Customization	Modification Degree	1)Programming 2) Customization 3) Parameterization
	Interface Documentation	1) Mandatory

Table 3 – COTS Specification

All said and done, components are the more economical choice for software organizations [8]. If developers could purchase 100,000 lines of code they could save 100,000 programmer days, thereby creating less expensive software. Not only that, according to Voas in T[18], if a world class programmer cost $500 a day, purchasing 100,000 lines of code would result in saving 5 million dollars.

The specification for a Commercial –Off –The Shelf is described in Table 3 above.

7 REUSABLE COMPONENTS

Reusability is an important engineering driver in the development of a component based system [26].

Before, we take a look at reusable components, let us first define reusability. According to Roger.S.Pressman [6], ”reusability of software is the extent to which a program or part of a program can be reused in other applications related to packaging and scope of the functions that the program can perform**”.** Jon Hopkins says that in the context of component based software engineering; reusability can refer to,” the ability to reuse existing components to create a more complex system” [26].

Category	Attribute	Value
Testing Level	Testing Type	1) Black Box Testing 2) White Box Testing
Reusability	Locatability	1) Difficult to locate
	Extensibility	2) Lower degree of extensibility
Usage	Functionality	Generic
Source	Origin	Internal
Type of Delivery	Packaging Type	1) Source code 2)Statically linkable binary library 3) Dynamic linkable binary library 4)Binary Component 5) Stand Alone Executable Program
Accessibility Level	Degree of Access	1)Greater degree of information hiding

Degree of Customization	Modification Degree	1)Programming 2) Customization 3) Parameterization
	Interface Documentation	1) Mandatory

Table 4: Reusable In-House Component Specification

Not to forget that a component can also be defined as "a reusable piece of software in binary form that can be easily integrated with other components with relative effort" (www.msdn.edu). So, even though the popular assumption is that all components are supposed to be reusable by virtue of definition, reality is very different. All components whether they are in-house or COTS may be reusable. However, the degree of reusability will vary from component to component. The higher the degree of reusability, the more generic and bulkier the component. The term reusable components can therefore be used to refer to those components which have a higher degree of reusability, and are more generic and bulkier in nature.

Using the above premise and Pressman's definition [6], a reusable software component can be defined as," software that already exists and that has been integrated a number of times in different software programs in the same application domain, with a high success ratio. Reusable components are more generic, as a result of which there are bulkier in nature". Using the above definition, we also infer that

1. An in-house component that adheres to the above definition is a reusable in-house component.
2. A COTS component that adheres to the above definition is a reusable COTS component.

Category	Attribute	Value
Testing Level	Testing Type	1) Black Box Testing
Reusability	Locatability	1) Difficult to locate
	Extensibility	2)Higher Degree of Extensibility
Usage	Functionality	Generic
Source	Origin	External
Type of Delivery	Packaging Type	1) Stand Alone Executable Program
Accessibility Level	Degree of Access	1)Greater degree of

		information hiding
Degree of Customization	Modification Degree	1) Customization 2) Parameterization
	Interface Documentation	1) Mandatory

Table 5 : Reusable COTS Specification

The TRUSTAD specification for reusable in-house and COTS component is given in Table 4 and Table 5 respectively.

According, to Michael Sparling [2], to ensure that an appropriate reuse occurs, a component has to be locatable, consumable and extensible.This also means that a component must have a complete specification, combined with some assurances that the component complies with the specification.

Again, there is a huge difference between a reusable in-house component and a reusable COTS component. This can be better explained with an example. Let us consider a consultancy that is going to develop an examination system say, for XYZ University. It is easier to design because the domain is static and we already have an idea of the number of departments, the nature of examination systems etc for XYZ university.. On the other hand, if the consultancy has to develop software that can be used for any kind of university, the task is daunting because of the difference in requirements and therefore the component has to be more generic.

If the reusable component was in-house, the organization could probably at least immediately tell the developers to make the required changes. On the other hand, if the reusable component was a COTS component, it would be difficult to upgrade it because of the difficulties mentioned in section 6.Not only that, from the developers point, writing software for reusable COTS component is the most difficult of all because the component has to be very generic.

Reusable software components can be simple like familiar push buttons, text fields, list boxes and scrollbars. ORACLE and Microsoft Office are example of more popular reusable COTS components.To summarize, all components may not be reusable; reusable components are reusable to varying degrees. In general, a reusable component will be more complex and will carry a bulkier code as compared to non reusable components. Full reusability, though may be a designer’s ideal goal, can rarely, be achieved.

8 THE TRUSTAD NOMENCLATURE

We considered the specifications for In-house, COTS, reusable in-house and COTS components using the seven dimensional vector (T, R, U, S, T, A, D) vector in the previous sector.Fig 2 provides a consolidated view. It can be seen that the TRUSTAD nomenclature enables us to clearly demarcate between various components.

From a perusal of the figure,

1. **All** software components satisfy criteria **1, 21** and **22**

1. .2.While **In-house** components satisfy criteria
2. **1, 2, 3, 5, 7, 9, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22 and 23**.
3. **COTS** components satisfy criteria **2, 3, 5, 7, 10, 12, 13, 14, 15, 17, 20, 21, 22 and 24.**
4. Similarly, **Reusable In-house** components satisfy criteria
5. **1, 2, 4, 6, 8, 9, 11, 12, 13, 14, 15, 17, 20, 21, 22 and 24**
6. **Reusable COTS** components satisfy
7. criteria 2,4,6,8,10,15,17,21,22,and 24.

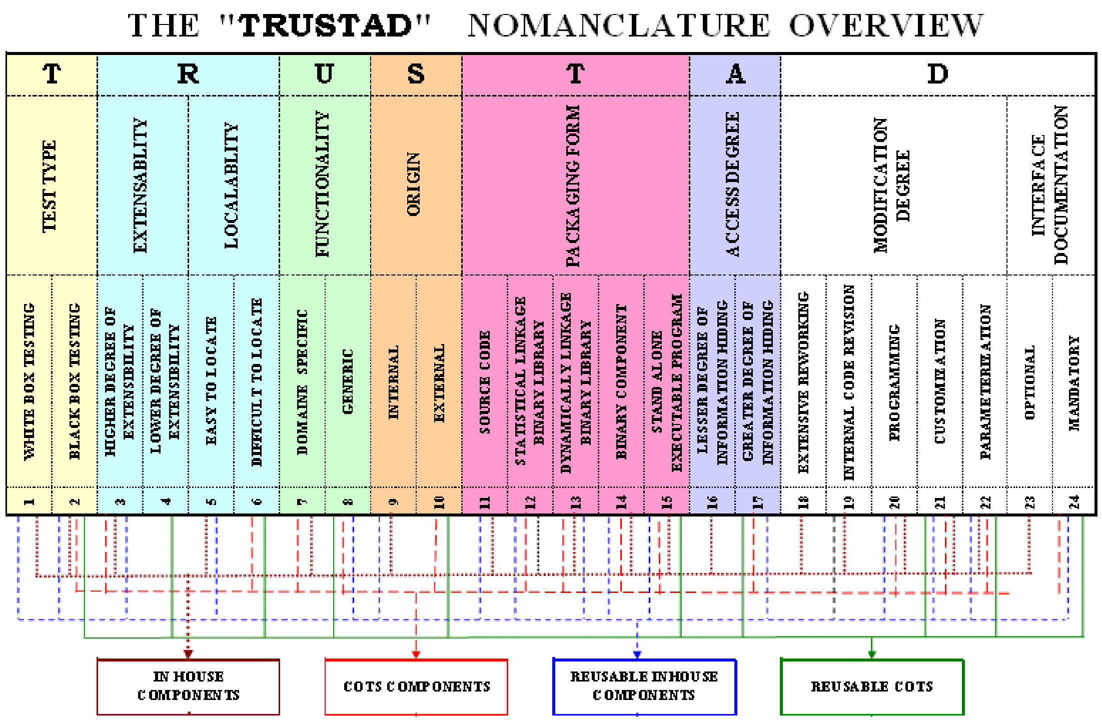

Figure 2 : The TRUSTAD Nomenclatur

9. CONCLUSIONS

Industry and the open market have had a significant impact on the development of component technology. A consequence of this situation is that component based software engineering uses concepts that are still not fully formalized and terms that are not clearly distinguished. This paper has made an attempt to define and clarify the differences between the terms software components, in-house components, COTS components and reusable in-house and COTS components.

We proposed that a software component can be represented by a seven dimension vector (**T, R, U, S,T, A , D**). Using the above proposal, specifications were written for in-house components, COTS components, reusable in-house and reusable COTS components. It could be seen that the values differ when it came to source, degree of customization, type of delivery, and the kind of testing possible. We attempted to define software components, in-house components, COTS components and reusable in-house and COTS components.

Further we tried to provide an answer for the question,” What is a reusable component?” We found that all components are not reusable. In fact, the degree of reusability varies from component to component and reusable components are more *generic* and *bulkier* in nature. We also concluded that the above seven dimension vector is sufficient to distinguish between in-house and COTS components. Finally, we consolidated the four specifications to arrive at our TRUSTAD component nomenclature classification.

Thus, this paper is just a small step in the direction of trying to distinguish between various component types - in-house, COTS and reusable. We invite the readers to offer their comments.

9 ACKNOWLEDGEMENTS

The authors record grateful thanks to late Dr.Peer .S. Mohammed, the Correspondent, Dr.V.M.Periasamy, the Principal and Dr.T.S .Rangasamy, the Dean Academic for the facilities provided. They would also like to express their thanks to Professor Manu Natarajan, Head of the Department of CSE for useful discussions.

REFERENCES

[1] Sparling M.: “Is there a Component Market”, www.cbd_hq.com/articles/2000

[2] Sparling M.: ”Lessons learned through six years of Component Based Development”, Communications of the ACM, 2002.

[3] J. Gao, K. Gupta, S. Gupta and S. Shim: “On building testable Software Components”, COTS based Software Systems, volume 2255 of LNCS, pages 108-121, Springer-Verlag, 2002

[4] M. Haddox, M. Kapfhammer, C. Michael: “An Approach to Understanding and Testing Third Party Software Components“, 2002 Proceedings of IEEE Reliability and Maintainability Symposium

[5] J. Offutt, S. Kamsokeat and W. S. Rivepiboon: ”Increasing Class Component Testability”, Research Project submitted to the Centre of Excellence in Software Engineering, Chulalongkorn University, 2003.

[6] Gao Jerry: “Testing Component Based Software”, Starwest 2000

[7] E. JWeyuker: ”Testing Component Based Software: A Cautionary Tale”, IEEE Software, 15(5):54-59, 1998

[8] J. Voas: “COTS Software: the Economical Choice”, IEEE Software ,1998

[9] C. Szyzysperski: “Component Software Beyond OO Programming”, Addison Wesley, 1998.

[10] Roger S. Pressman: “Software Engineering”, Third Edition

[11] Moriso Moriso and Marco Torchiano: ”Definition and classification of COTS”, ICCBSS 2004

[12] D. Carney, F. Long: ”What do you mean by COTS”, IEEE Software, March/April 2006, pp 83-86

[13] Crnkovic, Hnich, Jonsson, Kiziltan: ”Specification, Implementation and Deployment of Components”, Communications of the ACM, Oct 2002.

[14] Sathit Nakkrasae, Peraphon Sophasthit: “A Formal Approach for Specification and Classification of Software Components”, SEKE 2002

[15] William T. Councill: ”Third Party Testing and the Quality of Software Components”, IEEE Software, Vol.16, No 4, pp 55-57, July/August 1999.

[16] G. Heineman and W. Councill: ”Component Based Software Engineering: Putting the pieces Together”,Addison Wesley,2001

[17] B. Meyer: “The Grand Challenge of trusted Components“, In Proc.ICSE 2003, pages 660-667, IEEE 2003.

[18] Jeffrey Voas: “Certifying Off –the- shelf Componnets”, IEEE Software Press, 1998

[19] Jeffery Voas: ”Maintaining Component Based Systems”, IEEE Software Press , July/August 1998,Pgs 22-27

[20] Miguel Goulao,Fernando Brito: “The Quest for Software Components Quality”,COMPSAC 2002

[21] Brereton.P, David.B: “Component Based Systems: A Classification of Issues”, IEEE Software Press 2000, Pages 54-62

[22] “Technical Concepts of Component Based Software Engineering”, 2nd Edition, Technical Report, Carnegie Mellon University

[23] Kung Kiu Lau, Zheng Wang: ”A Taxonomy of Software Component Models”, Proceedings of EUROMICRO-SEAA ,2005

[24] Sun Microsystems, The BeanBuilder, http://bean-builder.dev.java.net/

[25] Sun Microsystems, Java 2 Platform,Enterprise Edition, http://java.sun.com/j2ee.

[26] Hopkins Jon:” Component Primer”, Communications of the ACM, Oct 2000/ Vol 43, No.10.

[27] Susan Eisenbach, Chris Sadler: “Reuse and Abuse”, Journal of Object Technology, vol 6, no 1, January-February 2007, Pages 130-167 http://www.jot.fm/issues/issue_2007_01/article5/

[28] Julia Küster Filipe: "A logic-based formalization for component specification", in Journal of Object Technology, vol. 1, no. 3, special issue: TOOLS USA 2002 proceedings, pp. 231-248. http://www.jot.fm/issues/issue_2002_08/article13/

About the authors

Sivamuni Kalaimagal - received B.E degree from the University of Madras, Chennai, India in 1997 and M.Tech degree from Pondicherry Central University, Pondicherry, India in 1998. She is a member of ISTE and has ten years experience in teaching. She is currently pursuing her doctoral program in computer science and her area of research is software components. She can be contacted at kalai1276@gmail.com.

Rengaramanujam Srinivasan - born in 1940 in Alwartirunagari, Tamilnadu, India, received B.E. degree from the University of Madras, Chennai, India in 1962, M.E. degree from the Indian Institute of Science, Bangalore, India in 1964 and Ph.D. degree from the Indian Institute of Technology, Kharagpur, India in 1971. He is a member of the ISTE and a Fellow of Institution of Engineers, India. He has over 40 years of experience in teaching and research. He is currently supervising doctoral projects in the areas of data mining, wireless networks, Grid Computing, Information Retreval and Software Engineering.