Flexibility of the ComBAD* architecture

N.H. Lassing, D.B.B. Rijsenbrij and J.C. van Vliet
Vrije Universiteit, Faculty of Sciences
De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands
tel: +31 (0)20 44 47769, fax: +31 (0)20 44 47653
e-mail: {nlassing, daan, hans}@cs.vu.nl

Key words: Software architecture, software frameworks, software quality, quality assessment, flexibility, adaptability, portability, reusability

Abstract: Software architecture is nowadays regarded as the first step to achieving software quality. The main task of the architect is to translate the quality requirements into a software architecture. An important step in this process is to assess whether the architecture actually satisfies these quality requirements. The purpose of this paper is to explore which architectural choices support flexibility and how flexibility can be assessed. To that end, we explored the ComBAD architecture, whose main objective is flexibility, investigated the architectural choices made and assessed whether flexibility was achieved. Such will not only increase our insight into flexibility in general, but particularly into the assessment of this quality attribute.

We use the term flexibility in the broadest sense of the word. It will be used to denote adaptability, portability and reusability. Adaptability can be regarded as flexibility in the narrow sense, portability as the flexibility to use a system in various technical environments and reusability as the flexibility to reuse part of a system in another system.

* The ComBAD architecture is a software architecture developed within Cap Gemini Netherlands. ComBAD stands for Component Based Application Development.

1. INTRODUCTION

Recently, there has been an increasing interest in software architecture. It is nowadays generally accepted that the software architecture has a major impact on the quality of an information system. An important step to achieving the desired level of quality is to evaluate the architecture. The software architecture analysis method (SAAM), presented in Bass *et al.* (1998), was developed with this in mind. SAAM is a scenario-based assessment method that is mainly used to compare the usability of two or more candidate architectures. We claim that SAAM could just as well be used to assess the quality of a single architecture, to evaluate its usability in a certain situation.

The purpose of this paper is to explore how the three elements of flexibility could be addressed in an architecture and how SAAM could be used to assess to what extent the desired level of quality for these attributes was achieved. To do so, we investigated the ComBAD architecture, whose main objective is flexibility in the broadest sense of the word, we described the architectural choices made and we assessed its adaptability, portability and reusability using SAAM.

The remainder of this paper is organized into four sections. In section 2 we discuss the ComBAD architecture, in section 3 we assess its quality and in section 4 we make some concluding remarks.

2. THE COMBAD ARCHITECTURE

The ComBAD architecture was developed within Cap Gemini Netherlands, a large software house. The architecture originates from a project called “Reuse”, whose main purpose was to explore the possibility of reusing domain-knowledge. This project delivered an approach for Component Based Application Development, named ComBAD, and a supporting architecture, the ComBAD architecture, whose main quality requirements were adaptability, portability and reusability. This paper will focus on the ComBAD architecture. The corresponding development approach will not be discussed.

The ComBAD architecture was not developed for a specific customer, but it was intended for a broad category of administrative systems. Though it may be used for other domains as well, it is probably more suitable in some situations than in others. We return to this point in the evaluation in section 3.

In the next sections, we give an overview of the ComBAD architecture and the architectural choices made to achieve the quality requirements. This

overview consists of two parts, a description of the ComBAD framework, given in section 2.1, and a description of the ComBAD application architecture, given in section 2.2. These descriptions provide a high-level view of the architecture.

2.1 The ComBAD framework

The quality attributes addressed by the ComBAD framework are portability and reusability. Portability is the quality attribute that indicates the ease with which an application can be moved from one technical environment to another (Delen *et al.*, 1992). In the ComBAD architecture, portability is addressed by using a layered architecture, in which an application is separated from its technical environment, the latter consisting of things like the database-management system used for storage and the protocol used for communications. This separation is achieved by introducing an intermediate layer between the application and its technical environment, which abstracts from the details of this environment. This intermediate layer is the ComBAD framework. The ComBAD framework also offers the type of support required by applications that conform to the ComBAD application architecture.

An instance of this framework is created for a specific development environment and it consists of a number of concrete and abstract classes, and a definition of the way the instances of these classes interact. An application can use a framework in two ways: (1) by inheritance of (abstract) framework classes and (2) by calling methods defined in the framework's interface (Lassing *et al*, 1998). The abstract classes of the ComBAD framework are treated in the next section when we describe the application architecture. In this section, we focus on the interface of the ComBAD framework.

The ComBAD framework provides a common interface to the technical environment by encapsulating access to the environment in a number of services. These services include object brokerage, object persistency, transaction management, notify management, logging and security, each of which is implemented by one component in the framework. The underlying assumption for this is that potential changes in the technical environment each impact just one service and, therefore, also one component. Object persistency, for instance, is implemented by the object-persistency manager, which encapsulates access to the database-management system (DBMS). The impact of changing the DBMS is now limited to this object-persistency manager. Figure 1 shows all of the services of the ComBAD framework. In section 3.3, where we evaluate portability, we assess whether these services encapsulate the environment entirely.

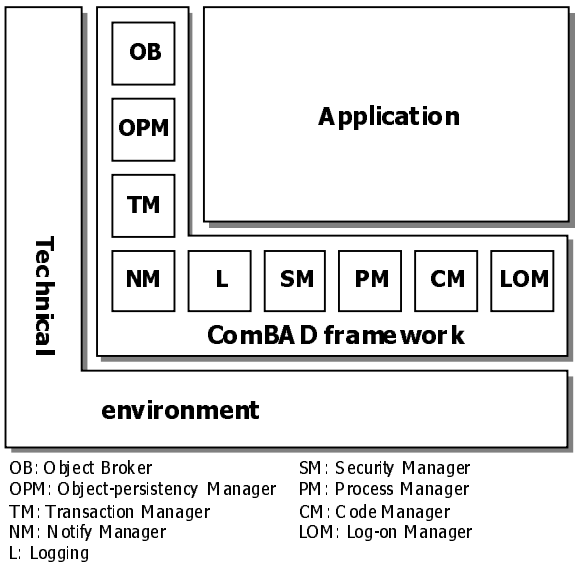

Figure 1. The layered architecture with the ComBAD framework and its services

The second quality attribute that the ComBAD framework addresses, is reusability. Reusability is much harder to achieve than portability, because it is more than a technical problem. Consider the following statement from van Vliet (van Vliet, 1993). He states that "a reusable component is to be valued, not for the trivial reason that it offers relief from implementing the functionality yourself, but for offering a piece of the right domain knowledge, the very functionality you need, gained through much experience and an obsessive desire to find the right abstractions". Apparently, reuse is only possible within a specific domain and we need a thorough understanding of this domain to determine its reusable elements. We define a domain as a well-defined area of application that is characterized by a set of common notions.

The ComBAD framework tries to address reusability in two ways. First, the framework itself can be reused. The domain in which this framework could be reused is the technical foundation of applications using the ComBAD architecture. Thus, the reusability of the ComBAD framework depends on the usability of the ComBAD architecture, which is the topic of the evaluation in section 3.

The second way in which the ComBAD framework addresses reusability is that it can serve as an environment for reuse of software components. This addresses one of the technical problems of reuse, namely architectural mismatch. Architectural mismatch occurs when the assumptions that a component and its environment make about each other are conflicting

(Garlan *et al.*, 1995). Frameworks reduce the risk of architectural mismatch, because they provide a known environment for components to operate in.

The ComBAD application architecture determines the types of components that can be used in the framework. They are included in the framework as abstract classes that implement the behavior that is common for these components. The actual components of the application are derived from these abstract classes.

Although frameworks address architectural mismatch, they are not a panacea for reuse. They do not relieve the developer from the painstaking process of finding the right components in a domain. However, they do provide support after the right components have been found.

2.2 The ComBAD application architecture

The quality attributes that are addressed by the ComBAD application architecture are adaptability and reusability. According to Bass *et al.* (1998), adaptability is largely a function of locality of change. This means that to increase adaptability, one should try to limit the impact of changes to a small number of components. On the other hand, we should try to limit the number of potential changes by which each component may be impacted.

In the ComBAD application architecture, adaptability is addressed by dividing an application into three layers: (1) the interface layer, (2) the processing layer and (3) the data layer. The interface layer handles the communication with the environment, consisting of users and other systems. The processing layer contains the application logic. Finally, in the data layer all data of the application is managed.

By separating an application into these layers, changes to the interface of the system can be limited to the interface layer, changes to the application logic limited to the processing layer and changes to the data limited to the data layer. However, this means that the number of potential changes that may impact each component is rather large. Therefore, it was decided to further divide the layers into components, as shown in Figure 2. The interface layer is divided into human interface components (or HICs), which handle a dialog with the user, and system interface components (or SICs) that communicate with other systems. The processing layer is divided into task-management components (or TMCs) that each implement (only) one function. And the data layer is divided into problem-domain components (or PDCs), which record data about a concept from the problem domain. Collectively, these components are called application components.

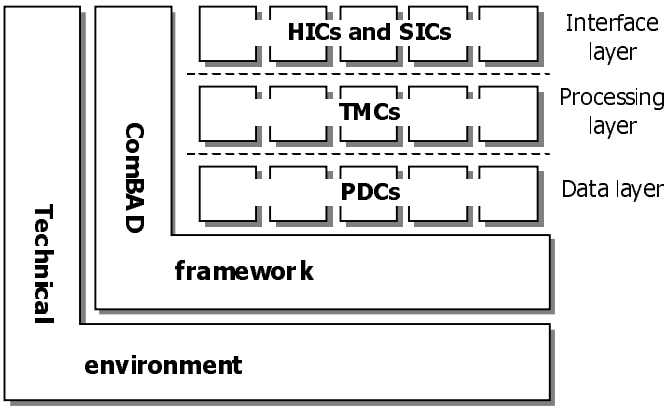

Figure 2. The ComBAD application architecture

This division can help us to limit the impact of changes to a few relatively small components. True locality of change is achieved for changes that affect the internals of one or more application components, but leave their interfaces intact. However, some changes not only affect the internals of one or more application components, but also the interfaces of some of them. This means that all dependent application components need to be changed as well. By restricting the dependencies between components, the impact of changes can be restrained. In the application architecture only top-down dependencies are allowed, i.e. a HIC or a SIC should only be dependent on one or more TMCs, a TMC on one or more PDCs and PDCs should be independent of other application components. Notify management is used to inform higher layers of events occurring in the lower layers.

Independence also affects reusability, because reusability demands that components are as independent as possible. Independence and, hopefully, reusability of PDCs is increased by prohibiting direct relations between them. This restriction reduces PDCs to stable building blocks that can be reused in other applications. We address the reusability of these components in section 3.4.

The ComBAD framework and the ComBAD application architecture very much depend on each other. First, the components of the application architecture use the services of the framework. For instance, the PDCs are accessed through the object broker and the PDCs use the object-persistency manager for storing themselves in a database. Second, the application components are derived from abstract classes provided by the framework. These abstract classes provide behavior common for each of these types of application components. Thus, the framework and the application architecture cannot be separated, they are highly intertwined.

3. ASSESSING THE QUALITY

To assess the quality of the ComBAD architecture we use the software architecture analysis method, SAAM (Bass *et al.*, 1998). This is a scenario-based method that consists of formulating a number of scenarios and evaluating the impact of each of them on the architecture. A scenario is a situation that can occur in the life of an architecture. Although SAAM was developed to compare two or more candidate architectures, we use it to assess the quality of a single architecture. The first step in the evaluation is to derive a number of scenarios from the quality requirements of the architecture. For example, from the quality requirement portability we can derive the following scenario: What happens when another DBMS is to be used? By formulating a number of these scenarios, we can make portability tangible, because they capture what we actually want to achieve with portability.

The next step is to evaluate the impact of these scenarios on the architecture. We classify the impact of a scenario into four discrete levels. At the first level, no changes are necessary, which means that the scenario is already supported by the architecture. At the second level, just one component of the architecture needs to be changed, but its interface is unaffected. At this level, we have true locality of change. At the third level more than one component is affected, but no new components are added or existing ones are deleted. This means that the structure of the architecture remains intact. At the fourth level architectural changes are inevitable, because new components are necessary or existing ones become obsolete. It is clear that one should seek to keep the level of impact as low as possible.

When we return to our example scenario, we see that this scenario necessitates a change in the object-persistency manager. Thus, this scenario has a level two impact. This means that we have locality of change for this scenario and that the architecture is portable with respect to the DBMS used.

We have created four categories of scenarios. The first two categories, which focus on adaptability, contain scenarios that are related to the requirements of the system. We have made a distinction between scenarios that address technical adaptability and those that address functional adaptability. The former consists of scenarios that explore the applicability of the architecture in situations with different technical requirements. The latter consists of scenarios that explore the effect of changes in the functional requirements.

The third category of scenarios concentrates on portability, which is evaluated by scenarios that simulate changes in the technical environment. The final category focuses on reusability. This category includes scenarios

that explore the use of elements of the architecture in other systems and architectures.

3.1 Technical adaptability

Technical adaptability is the flexibility of an application to incorporate changes to the technical requirements. The scenarios simulate the use of the ComBAD architecture in situations with diverse technical requirements. Note that the ComBAD architecture was not specifically developed for some of these situations. The architecture is usable in a situation when the scenario has an impact of level three or lower. The results of these scenarios are summarized in Table 1.

Scenario 1: *Which changes are needed when the architecture is to be used for secure applications?*

We assume that for secure applications a number of things are necessary. First, each user action should be authenticated and it should be possible to grant different levels of access to users (no access, read-only, full control, etc.). This is already supported by the ComBAD architecture, so it is unaffected. Second, the communication between clients and servers should be encrypted. Encrypted communication is not yet present in the architecture, but it could be added by changing one of the base classes for the application components. Finally, access to servers should be prohibited for unsecured hosts. This means that the log-on manager should be changed so that it inspects the network address of clients. Our conclusion is that using the architecture for secure applications necessitates changes to a number of existing components and, therefore, this scenario has a level three impact.

Scenario 2: *Which changes are needed when the architecture is to be used for real-time systems?*

The distinguishing features of real-time systems are deadlines and synchronization between different parts of a system (Laplante, 1993). These features are not supported by the ComBAD architecture. However, deadlines could be enforced by introducing something like a deadline manager into the framework, that makes sure that a systems responds within a certain period. Similarly, synchronization could be added by introducing a synchronization manager that makes sure that the different parts of a system operate in harmony. In addition, the division of applications into H/SICs, TMCs and PDCs is perhaps not usable for real-time systems. So, the impact of this scenario is architectural and it is classified as level four.

Scenario 3: *Which changes are needed when the architecture is to be used for ultra-reliable systems?*

In ultra-reliable systems both software and hardware are often replicated (Leveson, 1995). This redundancy makes sure that the system remains in working order after one or more services have failed. In addition, these systems could use voting, which means that the same operation is performed by two or more elements, and the end result of the operation is some kind of weighted average of the results of individual elements. Both redundancy and voting could be addressed by introducing one or more front-end servers that encapsulate the access to the other services. This has a major impact on the architecture and, therefore, the impact of this scenario is classified as level four.

Scenario 4: *Which changes are needed when a Web interface is created for an application?*

To make the system accessible from a Web browser, the human interface components (HICs) should be replaced with applets that can be viewed in a Web browser. Because the lower layers are independent of the HICs, they are unaffected by changes in the HICs. The HICs are the only components affected and thus the impact of this scenario is classified as level three.

Scenario 5: *Which changes are needed when the architecture is used for a system that uses workflow management?*

The ComBAD framework already has a process manager that controls which operations may be performed by the user in a certain situation. This component could be enhanced to support true workflow management. Since the process manager is the only component affected, the impact of this scenario is level two.

Table 1. Summary of the scenarios for technical adaptability

	ComBAD framework		Application				
Scenario	Archi-tecture	Components	Archi-tecture	HICs/ SICs	TMCs	PDCs	Imp. level
1	-	M	-	-	-	-	3
2	+	M	+	?	?	?	4
3	+	M	+	?	?	?	4
4	-	-	-	M	-	-	3
5	-	O	-	-	-	-	2

- = unaffected, + = needs to changed, O = one component affected, M = more components affected, ? = undefined

As expected, we see in Table 1 that the architecture is not directly usable in every situation. Using it for real-time or ultra-reliable systems necessitates major changes to the architecture. In the other situations, the architecture is usable, but some changes are necessary. These changes sometimes affect the framework and sometimes the application components. When the ComBAD architecture is used in an actual situation, more scenarios are probably needed to evaluate whether the right services are identified to encapsulate the expected changes to the technical requirements.

3.2 Functional adaptability

Functional adaptability is the ease with which changes in the functional requirements can be implemented. It is difficult to address the functional adaptability of an architecture, due to the absence of functional requirements. However, we are able to address the architectural aspects of changes to the functionality. To this end, we use scenarios that explore the effect of adding or deleting components from the application. The results are summarized in Table 2.

Scenario 1: *Which changes are needed when a problem-domain component (PDC) is added or deleted?*

When a new PDC is added, one or more elements in the higher layers should also be modified, for it does not make any sense to add a PDC without using it in one of the higher layers. When a PDC is deleted, the components in the higher layers that are dependent on it should be changed. The impact of this scenario can thus be classified as level three.

Scenario 2: *Which changes are needed when a task-management component (TMC) is added or deleted?*

A TMC is always invoked from the interface layer. Therefore, when a TMC is added, one or more H/SICs need to be changed to make use of this new TMC. Similarly, when a TMC is deleted, one or more H/SICs need to be changed to remove any references to the TMC. This scenario affects one TMC and at least one, but possibly more, H/SICs and the impact of this scenario can thus be classified as level three.

Scenario 3: *Which changes are needed when a human/system interface component (H/SIC) is added or deleted?*

The impact of this scenario is very small, because no other components are dependent on the H/SICs. In fact, the H/SIC that is added or deleted is the only component that is affected. Thus, this scenario has a level two impact.

Table 2. Summary of the scenarios for functional adaptability

Scenario	ComBAD framework		Application				Imp. level
	Archi-tecture	Components	Archi-tecture	HICs/ SICs	TMCs	PDCs	
1	-	-	-	M	M	O	3
2	-	-	-	M	O	-	3
3	-	-	-	O	-	-	2

- = unaffected, + = needs to changed, O = one component affected, M = more components affected, ? = undefined

From Table 2 we conclude that changes to the functional requirements do not affect the ComBAD framework. This means that the framework is entirely separated from the functionality of the application. And as expected, we observe that TMCs are unaffected by changes to the interface layer and that PDCs are unaffected by changes to either the processing layer or the interface layer.

3.3 Portability

At first sight, portability and technical adaptability very much look alike, but they are not the same. Portability is the ease with which a system can be adapted to changes in the technical environment and technical adaptability is the ease with which a system can be adapted to changes in the technical requirements. The scenarios in this category explore the effect of changes in the technical environment.

Scenario 1: *Which changes are needed when another database is used?*

This scenario was used in the introduction of this section. The object-persistency manager is the only element that is impacted. Thus, the impact of this scenario can be classified as level two.

Scenario 2: *Which changes are needed when another operating system is used for the client machines?*

The answer to this question is not unambiguous, because it depends on the programming language and the development environment used. First, if the application is written in Java, no changes should be needed, but other languages may cause major problems. Second, it is important which

development environment is used, because a number of development environments are able to generate and/or compile code for different platforms. This approach is taken in ComBAD, where the tools used can generate and/or compile code for multiple platforms. This solution lies outside the architecture, and the impact of this scenario can be classified as level one.

Table 3. Summary of the scenarios for portability

Scenario	ComBAD framework		Application				Imp. level
	Archi-tecture	Components	Archi-tecture	HICs/ SICs	TMCs	PDCs	
1	-	O	-	-	-	-	2
2	-	-	-	-	-	-	1

- = unaffected, + = needs to changed, O = one component affected, M = more components affected, ? = undefined

In Table 3, we observe that changes in the technical environment affect very few components of the ComBAD architecture. We notice that the application components are unaffected by our scenarios, which could indicate that the framework actually encapsulates access to the environment. However, there may be potential changes in the technical environment, not mentioned here, that have an impact above level two.

3.4 Reusability

We have chosen to assess reusability by scenarios that test the usability of ComBAD components in other situations, as well as the usability of other components within the ComBAD architecture. These scenarios focus on individual components, so it is not very meaningful to create a table that indicates which elements of the architecture are affected.

Scenario 1: *Can components that were not especially developed for the ComBAD framework, be used in applications built using the ComBAD architecture?*

The components that can be reused in these applications are mainly GUI-controls, like ActiveX-controls and Java Beans. However, because of the demands these components put on their environment, using them limits the portability of an application. Other components could be reused in these applications as well, if the components on which they depend are also included.

Scenario 2: *Can the application components be used in systems using another application architecture?*

The application components are usable in an environment that provides all of the framework services used by the component. This means that, theoretically, application components are reusable in another application, but it will require an enormous amount of work if they depend on more than a few framework services.

Scenario 3: *Can the object broker of the ComBAD framework be reused in systems using another architecture?*

The answer to this question is yes, provided all of the components upon which the object broker depends, being the transaction manager, the notify manager and the object-persistency manager, are included in the other architecture as well. However, this answer focuses on the architectural aspects only. Whether the object broker offers the right functionality in this situation is ignored.

Scenario 4: *Can application components be reused in other applications using the ComBAD architecture?*

Architecturally speaking, application components can be reused in other applications using the ComBAD architecture, provided the components on which they depend are also included. However, the reusability of a component also depends on whether it offers the right functionality. Within the ComBAD project, it was felt that the level of abstraction of the application components is too low. Therefore, a number of these components are grouped into packages, the same way Jacobson *et al.* (1997) address reusability. Whether these packages offer the right functionality can only be judged in an actual situation.

From these scenarios, we conclude that it is hard to assess the reusability of components, because it largely depends on the functionality they implement. From an architectural point of view, we may conclude that most components of the ComBAD architecture could be reused, but that this is easiest within the ComBAD architecture.

3.5 Evaluation of the assessment

In this section, we assessed the flexibility of the ComBAD architecture using scenarios. The assessment showed that the technical adaptability and

portability of a single architecture could be assessed quite well using scenarios, yet functional adaptability and reusability are harder to assess. The main difficulty of the assessment of functional adaptability of architectures is that functional requirements are lacking, which means we can only address the architectural aspects of changes to the functionality. Reusability is hard to assess in general, because the reusability of a component largely depends on whether it supports the right functionality, which can only be judged by a developer.

In addition, the assessment demonstrated that the flexibility of an architecture should always be related to the area of application. Although the assessment given in this section provides some general insight into the usability of the ComBAD architecture, one is unable to value the scenarios but in an actual situation.

4. CONCLUSION

The purpose of this paper is to explore how flexibility can be addressed in an architecture and how we can assess whether an architecture supports it. To that purpose, we have examined the ComBAD architecture. In the first part of this paper, we presented the architectural solution, which consists of the architectural choices made to address the quality requirements: adaptability, portability and reusability. We showed that in the ComBAD architecture portability and reusability are addressed by creating the ComBAD framework and that adaptability and, once again, reusability are addressed by the application architecture.

In the second part of this paper, we assessed the flexibility of the ComBAD architecture. To do so, we formulated scenarios for assessing technical adaptability, functional adaptability, portability and reusability. It turned out that assessment using scenarios of technical adaptability and portability of a single architecture is quite possible. However, functional adaptability and reusability proved to be hard to assess using scenarios, because we are considering an architecture, lacking functional requirements and actual application components.

The assessment demonstrated that the introduction of the ComBAD framework encapsulates changes to the technical environment from the application. In addition, we showed that the framework is unaffected by changes in the functional requirements. However, whether the services in the framework encapsulate the right technical mechanisms, could be a topic for further research. In addition, one should always remember that flexibility is a relative notion, which can only be valued in a particular context.

Our next step will be to investigate the architecture of an existing system to see whether we are able to assess its quality attributes, including functional adaptability and reusability of components. This way we hope to deepen our insight into the quality attributes and their assessment in software architectures.

ACKNOWLEDGEMENTS

This research is mainly financed by Cap Gemini Netherlands. We are grateful to Cor de Groot, Ad Strack van Schijndel and Guus van der Stap of Cap Gemini Netherlands for their time and comments. Guus van der Stap (e-mail: GStap@inetgate.capgemini.nl) can be contacted for more information about ComBAD.

REFERENCES

Bass, Len, Paul Clement and Rick Kazman (1998). *Software architecture in practice*. Addison Wesley, Reading.

Delen, G.P.A.J. and D.B.B. Rijsenbrij (1992) The specification, engineering and measurement of information systems quality. *J. Systems Software* **17**, 205-217.

Garlan, David, Robert Allen and John Ockerbloom (1995) Architectural mismatch: Why reuse is so hard. *IEEE Software* **12**, 6, 17-26.

Jacobson, Ivar, Martin Griss and Patrik Jonsson (1997). *Software reuse: architecture, process and organization for business success*. ACM Press, New York.

Johnson, Ralph E. (1997) Frameworks = (Components + Patterns). *Communications of the ACM* **40**, 10, 39-42.

Laplante, Phillip A. (1993) *Real-time systems design and analysis: an engineer's handbook*. IEEE Press, New York.

Lassing, N.H., D.B.B. Rijsenbrij and J.C. van Vliet (1998) A view on components. *Proceedings of the 9th International DEXA Workshop on Database and Expert Systems Application.* IEEE Computer Society, Los Alamitos, 768-777.

Leveson, Nancy (1995) *Safeware: system safety and computers*. Addision Wesley, Reading.

Shaw, Mary, and David Garlan (1996) *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Upper Saddle River.

Van Vliet, Hans (1993) *Software Engineering: principles and practice*. John Wiley & Sons, Chichester.