Motivational Issues in Creating Reusable Software Artifacts *

Gerhard Chroust, Christoph Hoyer
J. Kepler University Linz
Altenbergerstr. 69
A-4040 Linz
Austria
email: {gc,hoc}@sea.uni-linz.ac.at

Abstract

Despite all progress in technology and considerable publication effort, component based software development (CBD) seems to get a slower start than anticipated. To some extent this can be attributed to psychological ad motivational soft factors of software developers. Component based development has three essential players: the provider of components, the consumer (user) of components and an intermediatory responsible for storage, dissemination and marketing. In this paper we discuss soft factors from the viewpoint of the component provider, arranging them according to the various steps in the component provision process and analysing them using Maslow's Hierarchy of Needs.

1 Reuse: COTS and Components

"Work smarter, not harder!" Avoiding (unnecessary) duplication of work by reusing available artifacts is obviously a key factor to increased productivity [Christensen, 1994]. Reuse promises higher productivity and shorter time-to-market (a key postulate for today's software production) but also higher quality [Allen, 2001; Arnold and Frakes, 1993; Chroust, 1996; 2002; Woodman *et al*., 2001].

After a period of unsystematic, ad-hoc reuse by individuals or a small groups reuse has become a recognized and systematized field of research and application, documented in many books and publications [Allen, 2001; Cheesman and Daniels, 2001].

Re-use of ideas, concepts, architectural patterns [Booch, 1998], designs and other high-level constructs promises more return on investment than reuse of code, because such artifacts preserve previously acquired domain knowledge.

Despite all technological advances, publicity and lip service software components have not achieved the widespread use their proposers hoped for. We suspect that part of the reason for the slow uptake is caused by the psychological disposition of software engineers. In this paper we analyse soft factors (motivational issues and psychological preconditions) connected with the creation and provision of software components in a CBD-environment.

The rest of the paper is organized as follows. In chapter 2 we discuss the basic concepts of CBD. Chapter 3 introduces the importance of considering soft factors and Maslow's Hierarchy of Needs. The component provision process of CBD is shown in chapter 4 and the individual steps are discussed in relation to Maslow's Hierarchy of Need, followed by some overall soft factors relevant to the overall process (chapter 5).

2 The re-use life cycle

When speaking of reusable artifacts several types of prducing them may be distinguished:

Commercial off the Shelf (COTS) :
These are ready-made pieces of software which, in the fashion of a black box, can be integrated into other software systems to deliver a certain functionality without allowing access to their internal structure [Voas, 1998].

Components :
Components have to conform to the architecture of a *component model* which specifies the way they are to be build and integrated, and how they communicate with one another [Allen, 2001; Bachmann and others, 2000; Cheesman and Daniels, 2001].

Software patterns :
A Software Pattern is a *proven, non-obvious, and constructive solution to a common problem in a well-defined context, taking into account interactions and trade-offs (forces") that each tend to drive the solution into different directions. A Pattern describes the interaction between a group of components, hence it is a higher-level abstraction than classes or objects.* [Bruyninckx, 2002].

other reuseable artefacts :
Other intermediate results of a system design process (requirements specifications, designs, check lists, etc.) are also often useful to archive and to reuse later.

The life cycle of components undergoing reuse is depicted in Fig 1. It shows four ways of acquiring reusable components [Allen, 2001] which differ in the amount of pre-planning and marketing orientation.

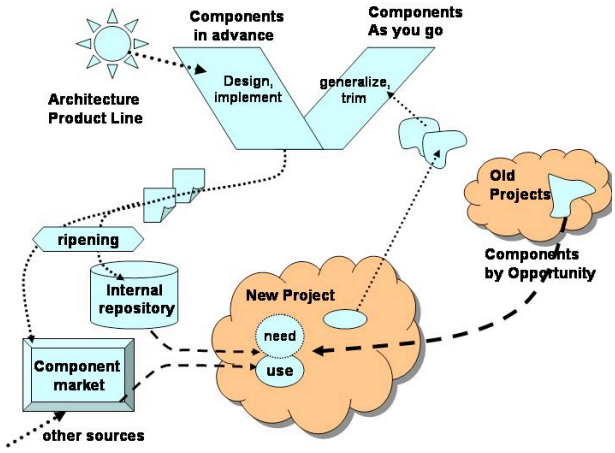

Fig. 1: Life Cycle of Reused Components

[Allen, 2001] identifies the following four ways to acquire components:

Components in Advance :
Components are planned based on expected need and demand, governed by a component model [Bachmann and others, 2000] and/or a product line model in a top-down, architecture driven way with special emphasis on quality [Woodman *et al*., 2001], later replaceability and marketability.

Components as you Go :
During system development certain modules/parts/concepts are identified as candidates for components, and after removing application specifics and adding potentially needed details ('homogenization') these components are made available, often after a certain 'ripening phase' (beta-versions) internally and/or on the free market.

Components by Opportunity :
When need arises old artifacts are taken, mostly due to the memory of a developer that "we had that already", on the fly adopted and used (often via cut and paste). Derogatively this is called the 'quarry' or 'scavenging' method. It is unsystematically and often 'quick and dirty'.

acquisition on the component market :
On the component market components may be acquired which did not originate in the own organisation.

3 The software engineer and the Hierarchy of Needs

For a provider of components (be it COTS or components on any development level) above four alternatives require a different attitude for developers and as a consequence yield different criteria for emotional satisfaction.

In [Chroust, 2002] and in [Chroust, 2003] we have described mostly demotivators for software engineers *as users* to follow the re-use paradigm, using COTS and/or software components of any sort. In this paper we will look 'at the other side of the fence': motivators and demotivators for *producing/providing* reusable artifacts (more or less for others):

Obviously the biggest driver for component providers would be a strong market demand. In the case of new paradigms, a market has to be *created* from scratch. Given the reluctance of many developers (i.e. potential customers of a component market) availability much precede and trigger the demand. This means that potential component providers have to have a motivation for producing components.

We therefore investigate the soft factors [McConnell, 2000; Lynex and Layzell, 1997; Enzenhofer, 2001; Chroust, 2002; 2003] involved in producing for a component market from the viewpoint of a component provider, with special emphasis on soft factors considering the soft factors in this transition period is of utmost importance.

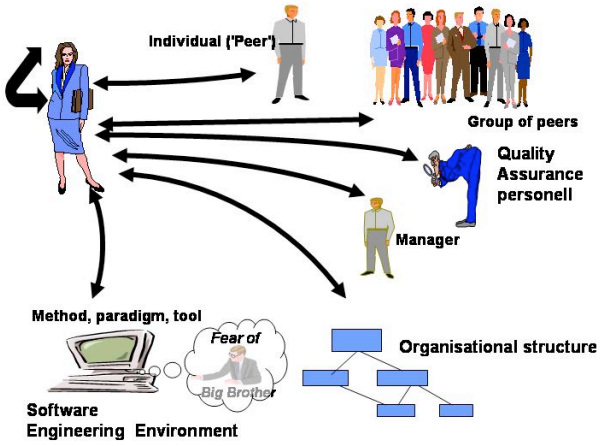

Fig. 2: Relations of an Individual

[Kunda and Brooks, 1999] emphasise that *Software systems do not exist in isolation. They are used in social and organisational contexts. Experience and many studies show that the major cause of most software failures is the people rather than technical issues [Curtis* et al.*, 1988] ... It is the people and culture of the organisation that determines how any system is used. ... human, social and organisational considerations affect software processes and introduction of software technology.*

Following [Chroust, 2002] we will use Maslow's Hierarchy of Needs to analyse the consequences of the multiple relations any individual is involved in, e.g with the peers, the managers, the quality assurance personnel etc. (Fig. 2). We will put this into contrast to the observations in [Chroust, 2002] where we considered the other side of the coin: engineers who want/have to use pre-existing components.

Maslow [Maslow, 1943; Boeree, 1998; Chroust, 2002] described a five-level hierarchy of needs (Fig. 3) with the assumption that lower level needs have to be substantially fulfilled before a strong interest in the higher levels is felt.

Fig. 3: Maslow's Hierachy of Needs

basic physiological needs (survival) :
At this level the individual is fighting for survival against an adverse environment, trying to avert hunger, thirst, cold etc. This level does not seem to have any relevance to our subject; software engineering is a save, non-endangering activity.

security (physical, economic, ...) :
On this level the individual's concern is about stability of the future and about a safe environment. Worries include job security, loss of status, loss of income, health etc.

social environment (community) :
Engineers usually have a strong identification with their status as 'engineers', and a strong attachment to their professional class. This implies certain standards and a certain way to look at work. This need is stressed by the observation [Glass, 1983] that software engineers develop a remarkable loyalty to their company.

recognition :
Individuals strive for receiving appropriate recognition and appreciation in the workplace, being recognized as somebody with 'something to say'.

self-fulfillment :
This level can be seen as the highest stage in the development of a person, drawing satisfaction from recognizing one's own contribution to some higher goal or endeavour and being able to realize one's full potential as a human being.

4 Providing software components

In this section we will discuss some of the motivators/demotivators for providing re-usable artifacts for the current and future software development based on the provision cycle shown in Fig. 4 We will not discuss general obstacles to paradigm change as described in many other publications ([Chroust, 2002; Enzenhofer and Chroust, 2001; Hoch, 1991; Greswell, 1989; Kuhn, 1970]). We restrict our discussion to the issue of providing of components in particular and relate them to the the levels in Maslow's Hierarchy of Needs, seen with the eyes and with the motivational situation of a producer of reusable software artefact, we can identify some soft factors. Fig. 4 shows the necessary steps [Chroust *et al.*, 1994; Chroust, 1996]:

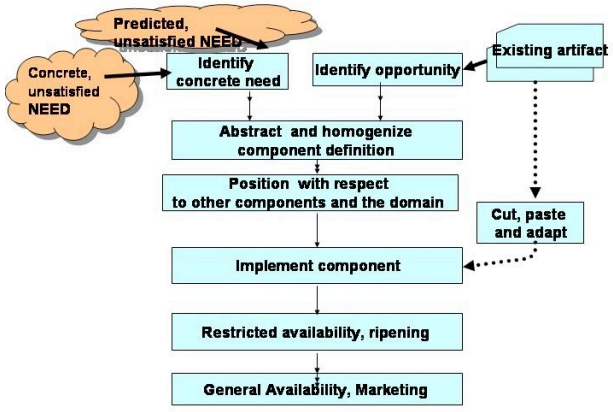

Fig. 4: Process Steps for Component Provision

4.1 Identify specific need

In general the need for a certain functionality will trigger the search for an existing component, which might identify a need to produce a *new* component. As indicated in Fig. 4 a 'quick and dirty' way is to cut and paste the needed functionality from some existing artifact.

With respect to Maslow's hierarchy, we see a window of opportunity for someone to

social environment, recognition, self-fulfilment :
The consequence of this work very much depends on the climate in the organisation and the personality: is the identification of a need seen as constructive and 'far-sighted' or is it considered as a nuisance - or even unnecessary. In general we expect to have a positive influence on the person, because he/she shows insight, far-sightedness and a crisp mind.

4.2 Identify opportunity

During the project often the reuse potential of a component is not recognized - only in hindsight one sometimes recognizes a potentially useful compoent.

social environment, recognition, self-fulfilment :
The similar arguments hold for this step, but based on more solid evidence and even the chance to prove the productivity gain

4.3 Abstract and homogenize component definition

A function selected for reuse must still be made into a component. The function must usually be augmented and modified by some additional functionality to be sufficiently generic. At the same time specifics for the original application (e.g. type of printer, ...) have to be removed. A subtle compromise between a module's broad applicability and

its efficiency must be made. As with any other marketable product there must be a compromise between the fully, potentially achievable functionality and the economics of the achievable. Engineers also have to live with 'good-enough-quality' (cf. [Bach, 1997]) as provided by the components and often cannot achieve 'best quality'. This is often difficult to accept emotionally by engineers.

security (economic) :
Our society is largely based on the notion of ownership which also extends to immaterial products like ideas and intellectual achievements ('intellectual property rights', IPR). Such ownership is protected and defended by copyright and patent legislation. Plagiarism is objected on moral grounds and often punished [Kock, 1999]. What is the division(?) of IPRs between the original creator of a piece of code and the later 'homogenizer', putting in ideas, largely surpassing the originators own views?

recognition, self-fulfillment :
This step needs persons with strong abstraction capabilities and imagination. Such persons are usually well accepted in a community and derive considerable self-fullfillment from such work.

4.4 Position with respect to other components and the domain

This is an essential step because it strongly influences the acceptability of a component, because users will evaluate the need in view of the assembly of all components. Too much overlap or remaining gaps are detrimental to a component's acceptance.

security (physical, economic, ...) :
Miscalculations and misinterpretations of the component market and the specific domain can have detrimental effects on the economic situation of the producing organisation and as a consequence also on the persons responsible.

social environment, recognition :
In order to be able to position the future components correctly, considerable contacts, with other developers, professional organisations are necessary, usually leading to a certain improved social standing.

4.5 Implement component

Few aspects specific to the implementation of components are discernible. The same situation as in any other software development project exists. Basically this step is the same as in any other environment. Difficulties arise from the fact that for components higher requirements for trust have to be applied if a component is intended to be used as a black box in another environment. Different environmental influences (different hardware, software and platforms) have to be considered.

security (physical, economic, ...) :
Two divergent trends can be seen here. Components being more restricted by rules and regulations (e.g. the component model) give a certain base security to the developer

self-fulfilment :
Due to the often stringent rules and regulations it is expected that developers will not feel free about their way of working and thus only enjoy a more limited amount of self-fulfilment.

4.6 Restricted availability, ripening

Due to the intended use *outside* of the original domain of definition there is a higher risk that the component does not perform as expected in all situations. Additionally being used as a component in a foreign environments requires a higher quality and thus establishing a higher level of trust in the component. The ripening step will also be a time of changes and modifications to the component.

security (physical, economic, ...) :
People usually dislike if their work is scrutinized by other people, cf. [Chroust and Lexen, 1998; 1999] because they feel threatened.

recognition, self-fulfillment :
If the component is successful, both recognition and the feeling of achievement will be the reward for the developer, perhaps even gaining over time the status of a 'great designer', as F.Brooks postulated for software professionals [Brooks, 1986].

4.7 General Availability, Marketing

No specific differences exist to the previous step, only that the effects (both positive and negative) are reinforced.

5 General Motivators and Demotivators

Besides the specific motivators and demotivators discussed in section 4, general ones exist, too.

5.1 Security

The need for security is threatened by numerous factors related to the introduction of new technology. There are a few intriguing aspects if considering the software engineer providing reusable artifacts.

job security: :
It has to be recognized that considerable know-how goes into a component. Know-how which originally was only located in the engineer's head. A certain danger of becoming redundant is lingering there. If one assumes, however, that the component will be a strategic component of other systems - even of systems alien to the original source, it is to be expected that the component will have to undergo modifications. And thus the originator of the component might ever become indispensable. Being the originator of a component, if it is successful, we be - rightly - as an expert in that domain.

compromising: :
As with any other marketable product there must be a compromise between the fully, potentially achievable functionality and the economic temporal achievable. Engineers also have to live with 'good-enough-quality' (cf. [Bach, 1997]) as provided by the components and often cannot achieve 'best quality'. This is often difficult to accept emotionally by engineers.

5.2 Social Environment (community)

intellectual property rights: :
Our society understanding of ('intellectual property rights', IPR), i.e. ownership on immaterial products like ideas and intellectual achievements. What part of the IPRs belongs to the creator of the original component and what part has the homogenizer?

change of work organisation: :
CBD needs a different work organisation [Kunda and Brooks, 1999], a rearrangement of areas of responsibility, including power and status, potentially causing upsetting an established social climate and well-established conventions.

isolation :
Developers of componentent could get the feeling that they are second class developers, having (to a large extent) work on and homogenize etc. conmponents who were invented by somebody else, they will be attacked if their modules are not delivered on the promised time or in the agreed upon quality etc.

5.3 Recognition

new guru-ship :
Components encapsulate and hide implementation details and domain know-how. The originator of components will own this knowledge and thus become an indispensable expert for this component. A strong motivator for an individual is recognition by the relevant social or professional reference group. For software engineers the reference group is the peer group of software engineers.

the CBD-water carrier: :
Component development is mostly concerned with making existing modules reusable, and not with creating new ones. Jobs in the 'reuse-unit' (similar to maintenance units [Basili, 1990]) might be considered to require less know-how and receive lower prestige, despite the fact that these jobs might require higher know-how and experience than the job to design a system from scratch.

isolation :
"Just delivering components" could be seen by management as a not too challenging job, endangering these professionals of being overlooked at promotions etc.

shifting of prestige: :
Successful CBD needs a change in organisation [Kunda and Brooks, 1999] which means that certain persons gain or loose influence, power and prestige. With respect to CBD - which needs a different mind set and also to some extend other abilities - other persons will become the centre of attention, a dramatic danger Where does one stand in the pecking order?

contempt: :
The inherent individuality of software development ('lone wolves') together with the multitude of different ways to solve the same problem has tempted many software developers into contempt of everybody else's work, if it is different from 'their way'. Thus the necessary recognition for the component developers could be missing.

5.4 Self-fulfilment

The ability to design "wonderful" systems is a strong motivator for software engineers. This feeling goes beyond recognition of peers - one *knows it oneself*. This makes it difficult for component developers to make their components accepted by other people work: *Not invented here!* [Campell, 2001], [Disterer, 2000]. CBD by necessity has to rely on external sources in terms of a global market.

goldplating: :
The feeling of self-fulfilment often cannot accept the knowledge that a component *should/must still be improved*, leading to endless effort in gold-plating a system before delivery (or even thereafter). This temptation becomes larger since the components are intended to be used "by the whole world".

creativity: :
The process of recognizing a need for a component plus the subsequent specification of the component is highly creative and therefor self-fulfilling.

6 Summary

Soft factors like motivations and psychological preconditions play a stronger role than expected even in a technical field like software engineering. These soft factors could account for the slow uptake of component based software development. In this paper we discussed some of the soft factors potentially inhibiting a faster acceptance of component based software development from the view point of a software component provider. We have shown the sources for components and the steps leading to a component on a 'market', be it an internal data base to take components from or a true externally accessible competitive market. Using the framework of Maslow's Hierarchy of Needs we identified important aspects of (non-)fulfilments of needs in the course of component based development projects.

References

[Allen, 2001] P. Allen. *Realizing e-Business with Components*. Addison-Weseley 2001, 2001. ISBN 0-201-67520-X.

[Arnold and Frakes, 1993] R.S. Arnold and W.B. Frakes. Software reuse and reegineering. *Arnold, R.S.(ed.): Software Reengineering, IEEE Computer Society Press, Los Alamitos, Calif. 1993*, pages 476–484, 1993.

[Bach, 1997] J. Bach. Good enough quality: Beyond the buzzword. *IEEE Computer vol. 30 (1997) no. 8*, pages 96–98, 1997.

[Bachmann and others, 2000] F. Bachmann et al. Volume ii: Technical concepts of component-based software engineering. Technical report, CMU/SEI-2999-TR-008, ESC-TR-2000-007, May 2000, 2000.

[Basili, 1990] V. R. Basili. Viewing maintenance as reuse oriented software development. *IEEE Software Jan. 1990*, pages 19–25, 1990.

[Boeree, 1998] C. G. Boeree. Abraham Maslow, Biography. http://www.ship.edu/~cgboeree/maslow.html (2001-11-18), 1998.

[Booch, 1998] G. Booch. Architectural patterns. http://www.rational.com/products/ whitepapers/390.jsp 2001-07-30, 1998.

[Brooks, 1986] F.P.Jr. Brooks. No silver bullet - essence and accidents of software engineering. *Kugler H.J. (ed.): Information Processing 86, IFIP Congress*, pages 1069–1076, 1986.

[Bruyninckx, 2002] H. Bruyninckx. Software patterns. http://www.orocos.org/patterns.html, Dez. 2002, 2002.

[Campell, 2001] J. Campell. Course on reuse: Impediments to reuse. http://www.cs.qub.ac.uk/~J.Campbell/ myweb/misd/ node8.html#section00840000000000000000 (excerpt, 2001-11-17), 2001.

[Cheesman and Daniels, 2001] J. Cheesman and J. Daniels. *UML Components*. Addison Wesley, 2001, 2001. ISBN 0-201-70851-5.

[Christensen, 1994] S.R. Christensen. Software reuse initiatives at lockheed. *Lockheed Horizons, Lockheed Corp., December 1994*, 1994.

[Chroust and Lexen, 1998] G. Chroust and H. Lexen. Software-inspections : Yes, but.... - improving the quality of software products. In *Hofer, S. and Beneder, M.: IDIMT'98, Proc. 6th Interdisciplinary Information Management Talks, Oct. 21-23, 1998, Zadov, Czech Republic, Trauner, Linz*, pages 227–240, 1998. ISBN 3-85320-955-6.

[Chroust and Lexen, 1999] G. Chroust and H. Lexen. Software inspections - Theory, new approaches and an experiment. In *Tyrell, A.: Euromicro 1999 IEEE Computer Press 1999*, pages 286–293, 1999.

[Chroust *et al*., 1994] G. Chroust, P. Grünbacher, and S. Hofer. Software 2001 – eine Vision und ein Weg. *Kepler Univ. Linz, Techn. Bericht für Softlab Österreich, Juni*, 1994.

[Chroust, 1996] G. Chroust. Software 2001 – ein Weg in die Wiederverwendungswelt. In *Lehner F. (ed.): Softwarewartung und Reengineering – Erfahrungen und Entwicklungen Gabler Edition Wissenschaft, Deutscher Universitätsverlag*, pages 31–49, 1996. ISBN 3–8244–6294-X.

[Chroust, 2002] G. Chroust. Motivational issues in component based software development. In *Trappl, R.: EMCSR 2002, Proc. European Meeting on Cybernetics and Systems Research, Vienna, April 2002*, pages 165–170, 2002. ISBN 3 85206 160 1.

[Chroust, 2003] G. Chroust. Software Komponenten - ungeliebte Kinder der Software-Ingenieure. In *Fiedler, G. and Donhoffer, D.: Mikroelektronik 2003, Wien*, pages 577–588. ÖVE Schriftenreihe, Nr. 33, 2003.

[Curtis *et al*., 1988] B. Curtis, H. Krasner, and Iscoe N. A field study of the software design process for large systems. *Comm ACM*, 31:11:1268–1298, 1988.

[Disterer, 2000] G. Disterer. Individuelle und soziale Barrieren beim Aufbau von Wissenssammlungen. *Wirtschaftsinformatik, vol. 42 (2000), no. 6*, pages 539–546, 2000.

[Enzenhofer and Chroust, 2001] W. Enzenhofer and G. Chroust. Best practices approaches in know-how and technology transfer methods for manufacturing smes. In *Proc. 27th EUROMICRO Conference, Sept 4-6, 2001, Warsaw*, pages 279–286. IEEE Computer Society, 2001. ISBN 0-7695-1246-4.

[Enzenhofer, 2001] W. Enzenhofer. *Best Practice Implementation of Advanced Information and Communication Technologies in Manufacturing SMEs*. PhD thesis, J. Kepler University, PhD-Thesis, Nov. 2001, 2001.

[Glass, 1983] R.T. Glass. *Software Runaways*. Prentice Hall 1998, 1983.

[Greswell, 1989] C.A.E. Greswell. Technology transfer of quality management tools. *Bennett K.H. (ed.): Software Engineering Environments - Research and Practice Ellis Horwood Books in Information Technology,*, pages 289–298, 1989.

[Hoch, 1991] D.J. Hoch. Management von Technologie-Diskontinuitäten in Informatik-Projekten. *Elzer P. (ed.): Multidimensionales Software-Projektmanagement AIT-Verlag München*, pages 153–179, 1991.

[Kock, 1999] N. Kock. A case of academic plagiarism. *CACM, Vol. 42 (1999), no. 7*, pages 94–104, 1999.

[Kuhn, 1970] T. Kuhn. *The Structure of Scientific Revolutions*. Chicago Univ. Press 1970, 1970.

[Kunda and Brooks, 1999] D. Kunda and L. Brooks. Human, social and organisational influences on component-based software engineering. In *ICSE 99*, 1999. http://www.sei.cmu.edu/cbs/icse99/ papers/19/19.htm.

[Lynex and Layzell, 1997] A. Lynex and P.J. Layzell. Understanding resistance to software reuse. *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP '97) 0-8186-7840 IEEE 1997*, 1997. see andy.lynex<at>umist.ac.uk.

[Maslow, 1943] A.H. Maslow. A theory of human motivation. *Psychological Review, 50 (1943), also http://psychclassics.yorku.ca/ Maslow/motivation.htm (2001-11-18)*, pages 370–396, 1943.

[McConnell, 2000] S. McConnell. Quantifying soft factors. *IEEE Software vol. 17(2000), no. 6*, pages 9–11, 2000.

[Voas, 1998] J.M. Voas. Certifying off-the-shelf software components. *IEEE Computer, June 1998*, pages 53–59, 1998.

[Woodman *et al*., 2001] M. Woodman, O. Benedictsson, B. Lefever, and F. Stallinger. Issues of CBD product quality and process quality. In *4th ICSE Workshop: Component Certification and System Prediction, 23rd Int. Conf. on Software Engineering (ICSE), IEEE Computer Society, 2001*, 2001. ISBN 0-7695-1050-7.