Reuse with Software Components – A Survey of Industrial State of Practice

Rikard Land[1], Daniel Sundmark[1], Frank Lüders[1], Iva Krasteva[2], Adnan Causevic[1]

[1]Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden
[2]Faculty of Mathematics and Informatics, Sofia University, Sofia, Bulgaria
{rikard.land, daniel.sundmark, frank.luders, adnan.causevic}@mdh.se, ivak@rila.bg

Abstract. Software is often built from pre-existing, reusable components, but there is a lack of knowledge regarding how efficient this is in practice. In this paper we therefore present qualitative results from an industrial survey on current practices and preferences, highlighting differences and similarities between development with reusable components, development without reusable components, and development of components for reuse. Component reuse does happen, but the findings are still partly disappointing: currently, many potential benefits are not achieved. Still, the findings are encouraging: there are indeed good, reusable components properly verified and documented, and mature organizations who manage to reuse these components efficiently, e.g. by leveraging the previous component verification. We also find that replacing one component for another is not necessarily complicated and costly.

1 Introduction

The paradigm of component-based software engineering (CBSE) has a number of perceived benefits [1] [2]: components may be developed independently of each other and interact only through explicit interfaces, which open up the possibility for component reuse in new contexts. It provides a framework for defining architectures and facilitating ease of integration, when using pre-existing components as well as in a top-down design decomposition system development [3]. By selecting pre-existing components that have been proven in use and enhanced over time, it would be possible to construct high quality systems more rapidly than ever. Moreover, research is progressing towards the vision that system behaviour can be predicted from component behaviour [4] [5], which would make reuse even more attractive, as the consequences of selecting a particular component would be known in advance.

However, in practice, software reuse through components is difficult and not entirely successful, for several reasons: first, components do not always live up to the expectations, partly because it is inherently extremely difficult to verify a component without a context. Second, it is seldom easy to exchange a component for another; even though (part of) the interface is identical or similar. Thus, at least some of the development time saved through reusing a component needs to be spent in the selection, evaluation, and verification of components, and explicit management of the

relationships with component vendors. This typically also leads to vendor lock-in, and reuse is thus often degraded to only an initial event in a system's history.

We set out to study the state of the practice in the following general software development activities, from a software component reuse perspective: *requirements elicitation and customer interaction*, *design and implementation*, *verification*, and *component selection and evaluation.* For this purpose, we constructed a web-based survey and invited organizations reusing and integrating existing software components, as well as organizations not reusing components, and component builders, as respondents. Focus was on the technical staff (developers, testers, architects, etc.) This paper presents the results of this survey, thereby providing an insight in how well CBSE supports software reuse in current practice, how and to what extent components are verified in isolation, and how component users test and evaluate components before selecting them.

There are two main research questions reflected in the structure of this paper: first, in Section 4, we investigate whether there are any differences in how development activities are performed, depending on whether software development include the reuse of components or not. Developers of components for reuse are included as a third group in this comparison. Then, in Section 5, we investigate how *component selection and evaluation*, is performed by projects developing software (partially) by integrating reusable components.

First however, in Section 2 we describe the background, and in Section 3 we present the research method used to perform and analyse the survey. Section 6 concludes the paper and presents ideas on future work.

2 Background and Related Work

Although software reuse has some potential benefits, practice has shown a great many challenges, and not only technical aspects must be mastered. Any serious software reuse attempt must permeate the organization and allow existing processes and practices to be modified [6].

Other empirical studies of software reuse have been conducted (see e.g. [7] for a review) and even some focusing specifically on reuse with components [8] [9]. The study presented here adds to this body of work by investigating some specific questions, in particular related to verification and component selection. Other related work is referred to in context throughout the rest of the paper: Section 4 describes existing approaches to requirements and customer interaction [10] [11], design and implementation [11] [12], and verification [3] [13] [14], which has a bearing on component reuse. Section 5 relates to literature with suggested methods for Off-the-Shelf (OTS) component selection and evaluation [15] [16] [12] [17] [18] [19] [20] [21] [22]. In these sections, we describe suggested methods, practices, and previous observations found in literature, and relate our empirical survey results with them, to investigate the extent to which suggested guidelines etc. are adopted in practice.

3. Research Method

To study how the component-based software paradigm support reuse in practice, we constructed a web-based questionnaire. Invitation emails were sent to companies that were part of our joint research projects such as FLEXI[1] and NESSI[2] , among others. We thus received a total of 93 responses, 30 of which seem to have quit the questionnaire after providing only some background information. We believe the main reason is that they perceived the questionnaire would take too long time, and we cannot know if this poses a particular threat to validity, i.e. if some particular types of answers were thus systematically excluded.

However, since the respondents are anonymous we cannot know how many organizations these represent. Also, as we sent the invitation to participate to some email lists, and encouraged every recipient to further spread the invitation we can neither know the response frequency, nor exactly which organizations are represented. Hence, during any statistical treatment of the data we must bear in mind the limitations imposed by this type of convenience sampling to the external validity of the results. More information about the questionnaire, as well as all data, is available as a technical report [23].

In much of our analysis, we explore any differences between development *with* reusable components, development *without* reusable components, and development of components *for* reuse. These three groups are defined as illustrated in Fig. 1, based on three specific questions in the questionnaire: we consider two complementary subsets of development projects: *with* or *without* reusable components. Orthogonal to this division, we also consider projects developing components *for* reuse (which we study, as indicated in the figure), and projects developing (non-reusable) products and systems used by end users.

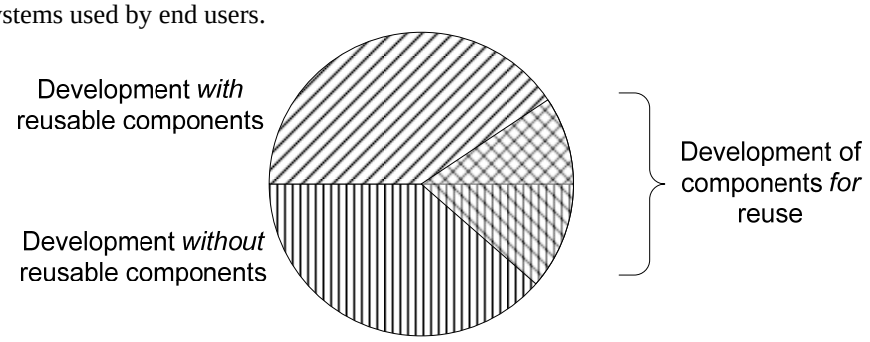

Fig. 1. Groups of respondents

For each respondent, based on the responses to some mandatory initial questions in the questionnaire, some later sections of questions were shown or hidden. As this caused the number of respondents to vary between sections, the number of respondents in each survey section is specified in Table 1, both per group and the total

[1] http://www.flexi-itea2.org/
[2] http://www.nessi-europe.com/

(which is the sum of the first two columns, i.e. the groups representing development with and without reusable components).

Table 1. Number of responses in each respondent group for each section in the survey

Development... / Survey Section	...*with* reusable components	...*without* reusable components	...of components *for* reuse	Total
Agile practice preferences	32	18	8	50
Testing	24	12	6	36
Component development	8	5	8	13
System development with reusable components	29	0	5	29
System development	25	0	6	25

4 Development *with, without,* and *for* Reuse

In this section, we analyze the activities *requirements elicitation and customer interaction*, *design and implementation*, and *verification*. In particular, we explore the differences, if any, between development *with* reusable components, development *without* reusable components, and component development *for* reuse.

4.1 Requirements Elicitation and Customer Interaction

Interaction with customers and feedback [11] affect how requirements are formulated, how fixed they are, and how often deliveries are made. Generally, our results show that regardless of the level of component reuse in development, incremental delivery is a widespread practice, but requirement handling and collection of customer feedback varies between development of, with, and without reusable components.

Regular interaction. For development without reuse, regular interaction between developers and customers/business people is in general encouraged by management, while for development with and for reuse, there is no consensus. However, there is a consensus among the respondents that they would like such regular interaction to be increased.

Changing requirements. For development with reusable components, there is a slight tendency to discourage customers from changing requirements once they are specified. For development without reusable components, the tendency is the opposite: customers have more possibilities to change their requirements. A possible explanation is that when a decision has been made to use a reusable component, requirement changes may have a larger impact on the existing design [10] [12]. However, both groups seem to be dissatisfied with the current state: respondents in the development with reusable components group would like to allow their customers to change their requirements, while respondents in the development without reusable components think customers should be allowed to change less. For the above questions, the development of components for reuse group provides answers without any clear preferences.

Incremental delivery. In all groups, the general practice is to deliver software to customers incrementally, and all respondents think this practice should be even more emphasized. All groups in general also provide users with early versions (alpha/beta) of the software, but this tendency is stronger among system development than component development for reuse. One interpretation of this difference is that it more useful feedback can be collected from end users using an incomplete or buggy user interface application, than from component users using an unreliable component.

Delivery of source code. Sometimes, the software is delivered as source code, and sometimes in binary format, without any particular tendency or any difference between the groups. This may indicate that the domain determines what is convenient, rather than for example different types of business relationships in the groups, or any difference in the desire to keep the implementation secret.

Customer feedback. In development of systems for end users, the respondents almost uniformly state that end customer feedback is collected and evaluated through different mechanisms. This can be contrasted to development of components, where the respondents are more varied in their responses, but still with a slight overweight in support for this practice. One partial interpretation is that for at least reusable components developed for the mass-market, the distance to customers is large (although this distance can be decreased: we are aware of one COTS vendor which presents the current state to their key customers in web conferences every second week, and allow interaction in these virtual meetings).

4.2 Design and Implementation

This section reports on some findings related to design and implementation from the perspective of the three groups defined above. Our findings point out that incremental design and coding is a preferred practice among the respondents, but also that there are differences between the preferred and the actual practice.

Interleaving of design and programming. The responses are very varied as to what degree programming should be allowed to start before design is completed. The current practice varies across the scale for all groups of respondents, although those doing development without reuse has a slight tendency towards being more permissive of starting programming early. When asked about their preference, all three groups are less permissive when compared to the current practice, although this is not the case for each individual respondent.

Incremental design and coding. This is often viewed as a good way to discover design problems early and to get early customer feedback [11]. Our findings show that it is widely used in current practice, independently of whether development is done with, without or for reuse. When asked about their preference, the respondents unanimously agreed that the incremental approach is desirable.

Return on investment of designing components for maintainability. The group of respondents representing development of software components for reuse unanimously agreed that if enough efforts for building a good and maintainable design of a component are not spent in advance, the cost of change for a component is really high. Respondents outside this group were not asked this question.

Redesigning component-based systems. Design lock-in has been identified as a potential side-effect of building systems from pre-existing software components [12]. However, the majority of respondents agreed that redesigning a system is not a big issue when building a system out of components. The group representing development without software components were not asked this question.

4.3 Verification

Ease of verification is one of the main arguments for software reuse through components [3] [14]. The main idea is that components that have been verified in previous settings and deployments will not require as much verification effort as software developed from scratch. Such savings would be highly relevant, since verification is widely known to consume significant portions of the resources in software development projects [13]. In this section, we investigate system and component verification from the perspective of current practice in software development with, without and for reuse.

General opinions. Regardless whether the system is built with or without reusable components, most respondents find themselves having less time for testing than they would like to. Looking at the ideal verification practices, in the eyes of the respondents, unit testing still has a high degree of preference. Moreover, respondents generally feel that both functional black-box testing and testing based on code analysis should be increased compared to current practice, and functional black-box testing is preferred over testing based on code analysis.

Unit testing and component testing. In system development with and without reuse, most respondents report a high level of use of unit testing. The same goes for functional black-box testing of components. This trend is even more apparent when looking at functional black-box testing on system-level. Answers are similar for performance and security testing, but we feel that these types of testing are too domain-specific to consider generally. In component development for reuse, both functional black-box testing and testing based on code analysis (e.g. statement or path coverage) are present in some projects. For all these verification methods, there is a noticeable difference between the current practice and the perceived ideal level of usage, which in general is significantly higher.

Integration testing. To a large extent, respondents find themselves in projects that allow code changes during integration testing. Interestingly, respondents developing systems with reusable components find this less problematic than those developing systems without reuse. In addition, the respondents do not consider it easier to test systems built out of reusable components which are previously tested in isolation, than to test systems built without reuse.

Testing of documentation. In all groups, testing of documentation is something that is perceived to be largely neglected, and most respondents, except those developing components *for* reuse, would like a significant increase in this practice. However, out of the 8 developing components for reuse, only 2 explicitly agree that the documentation provided with the components is sufficient for the needs for the component users.

In-house vs. Subcontracted vs. OTS. Among the respondents, there are stronger explicit demands on the documentation and verification of subcontracted components compared to the documentation and verification of in-house or OTS components. Component creators and component users both think that the current state of documentation and verification fulfil the needs of component users, for subcontracted and OTS components, but not for in-house components. This is also reflected in a stronger dissatisfaction with the documentation and verification of in-house components, compared to that of subcontracted or OTS components.

One possible explanation for this is that the distance from a subcontractor or OTS vendor to the component user is greater, and also that the amount (and quality) of documentation is regulated by contracts (for subcontractors), or implicitly required in order to have an attractive product (for OTS vendors). Whatever the reason, this points us in a direction where component reuse could be improved by providing more efficient and practically useful documentation.

5 Component Selection and Evaluation

In this section, we describe the current state of practice concerning the selection of reusable components to use during software development, and the challenges of evaluating reusable components in a system context.

5.1 Component Selection

The commonly suggested practice for OTS selection is to first filter away many component candidates in a high-level evaluation phase, based on information and documentation *about* the components, and only later perform a prototyping hands-on evaluation of a final few components by writing test cases and create prototypes [15] [16].

Roles involved in component selection. The survey responses indicate that in some projects, only the development unit is involved in the component evaluation and selection process, while other projects heavily involve customers or internal staff with a responsibility to know the market and customers. Although it is true that some components are not directly visible to customers and end users, more often than not, the decision to use a specific component does have a business impact; it may for example strongly affect the possibilities for future extensions of the systems [12] [17]. Thus, it appears that, in some companies, the current state of practice needs to be improved.

Interleaving system requirements elicitation and component selection. The respondents tend to formulate requirements on components fully prior to evaluation and selection. However, they generally find it difficult to break down system requirements to component requirements. This indicates that many organizations have not yet implemented the practice [15] to interleave component selection with the requirements elicitation process, as suggested by e.g. the methods PORE (Procurement-Oriented Requirements Engineering) [18], CRE (COTS-Based Requirements Engineering) [19] and CARE (COTS-Aware Requirements

Engineering) [20]. However, as said, the majority of the respondents assert that customers or business people are involved during component selection and evaluation. One interpretation is therefore that requirements elicitation and component selection and evaluation are often in practice interleaved, albeit not formalized as a process.

5.2 Component Evaluation

Prototyping evaluation. After some initial, high-level evaluation, based on information about OTS components (or existing knowledge of the potential components) [8], the suggested practice is to create prototypes, or simulate the system's usage of the component through testing [15]. There are two main goals for this: To examine technology or architecture [15] [16]; the survey results clearly show this type of prototyping activity is widely performed in practice. To evaluate component assemblies (rather than individual components) [15] [16] [21] [22]; the result varies with no clear tendency.

Usage of provided test cases. Our responses vary concerning whether test cases provided with the components are used to evaluate them. The respondents who use test cases provided with the components report that they also develop their own test cases for components in order to evaluate them, and surprisingly, those that do not use test cases provided with the components do not write their own test cases. Even more surprising is perhaps that this is true not only for subcontracted or in-house developed components – where one could expect the detailed functionality, level of quality, and responsibility for quality assurance to be specified by contracts – but also for OTS components.

Insufficient evaluation. High-level component evaluation and prototyping evaluation complement each other; however, if the components to select from are known, it may be sufficient to do a brief hands-on evaluation in the new context [8], which could partly explain that some of our respondents do not evaluate components prior to selection. However, some of the respondents who do not test their components believe testing is more efficient than documentation (this is true also for all respondents who do use test cases), which makes us lean towards the following conclusion: there are organizations and projects where OTS components are selected without proper evaluation – and that they are aware of this. However, there are also indeed organizations that perform systematic evaluation of OTS components.

6 Conclusion and Future Work

This paper presented an empirical, qualitative study of reuse with software components. Our data indicate that reuse of components does not make design decisions as permanent as might be feared. The impact of requirements changes are inconclusive. Regarding verification, the general opinion in our study is that it is not done to a sufficient extent, independent of component reuse. Separate verification of reusable components in isolation does not in general make system verification or

component evaluation easier. Known good practices for component selection and evaluation are implemented in some organizations but not all.

In conclusion, as for the current state of the practice of component reuse in industry, we can claim that components are as a matter of fact built for reuse, and those components are in fact being reused. The main reasons (which we *have not* studied) are probably those of cost and time for system development: through component reuse systems can be built cheaper and faster. However, some other potential benefits (which we *have* studied) are not in general experienced: in particular system verification is not necessarily made easier, and requirements engineering, and ultimately the ways system developers interact with their customer, need to change further than is the case in general today. Nevertheless, our study clearly shows that there are organizations where these benefits are indeed experienced, but this is apparently hard to achieve without explicit attention and effort. Further research includes studying the organizations which manages component reuse the best in order to identify good practices and how to implement them in different circumstances. Many such practices and potential benefits are already known, but are, according to our results, not yet widely adopted in industrial practice. As this generally confirms previous studies, it is useful as it adds to the body of knowledge and may provide additional insights.

Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS, the Bulgarian Ministry of Education and Science, and FLEXI. Thanks also to all the questionnaire respondents and the people who have been involved in earlier phases of this research.

References

1. Szyperski, C.: Component Software 2nd edn. Addison-Wesley (2002)
2. Wallnau, K., Hissam, S., Seacord, R.: Building Systems from Commercial Components. Addison-Wesley (2001)
3. Crnkovic, I., Chaudron, M., Larsson, S.: Component-based Development Process and Component Lifecycle. In : International Conference on Software Engineering Advances (ICSEA'06), Tahiti (2006)
4. Hissam, S., Moreno, G., Stafford, J., Wallnau, K.: Packaging Predictable Assembly with Prediction-Enabled Component Technology., Pittsburgh (2001)
5. Land, R., Carlson, J., Larsson, S., Crnkovic, I.: Towards Guidelines for a Development Process for Component-Based Embedded Systems. In : Workshop on Software Engineering Processes and Applications (SEPA), Yongin, Korea, vol. LNCS (2009)
6. Karlsson, E.-A.: Software Reuse : A Holistic Approach. John Wiley & Sons Ltd. (1995)
7. Mohagheghi, P., Conradi, R.: Quality, Productivity and Economic Benefits of Software Reuse: A Review of Industrial Studies. Journal of Empirical Software Engineering 12(5),

471-516 (2007)

8. Li, J., Torchiano, M., Conradi, R., Slyngstad, O., Bunse, C.: A State-of-the-Practice Survey of Off-the-Shelf Component-Based Development Processes. In Morisio, M., ed. : ICSR '06, Torino, pp.16-28 (2006)
9. Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O., Morisio, M.: Development with Off-The-Shelf Components: 10 Facts. IEEE Software 26(2), 80-87 (2009)
10. Cooper, K.: Can Agility be Introduced into Requirements Engineering for COTS Component Based Development? In : International Workshop on Software Product Management (IWSPM) (2006)
11. Beck, K.: EXtreme Programming EXplained: Embrace Change. Addison Wesley (1999)
12. Krasteva, I., Branger, P., Land, R.: Challenges for Agile Development of COTS Components and COTS-Based Systems – A Theoretical Examination., Funchal, Portugal (2008)
13. Tassey, G.: The Economic Impacts of Inadequate Infrastructure for Software Testing. (2002)
14. Aoyama, M.: New age of software development: How component-based software engineering changes the way of software development. Proceedings of International Workshop on Component-Based Software Engineering. (1998)
15. Land, R., Blankers, L., Chaudron, M., Crnkovic, I.: COTS Selection Best Practices in Literature and in Industry. In : Proceedings of 10th International Conference on Software Reuse (ICSR), Beijing, China (2008)
16. Oberndorf, P., Brownsword, L., Morris, E., Sledge, C.: Workshop on COTS-Based Systems. (1997)
17. Krasteva, I., Land, R., Sajeev, A.: Being Agile when Developing Software Components and Component-Based Systems – Experiences from Industry. In : EuroSPI, Madrid, Spain (2009)
18. Maiden, N., Ncube, C.: Acquiring COTS Software Selection Requirements. IEEE Software 15(2) (1998)
19. Alves, C., Castro, J.: CRE: a systematic method for COTS components Selection. In : Proceedings of the XV Brazilian Symposium on Software Engineering (SBES), Rio de Janeiro (2001)
20. Chung, L., Cooper, K.: Defining Goals in a COTS-Aware Requirements Engineering Approach. Systems Engineering 7(1) (2004)
21. Burgués, X., Estay, C., Franch, X., Pastor, J., Quer, C.: Combined Selection of COTS Components. In : International Conference on Component-Based Software Systems (ICCBSS), vol. LNCS 2255, pp.54-64 (2002)
22. Bhuta, J., Boehm, B.: A Method for Compatible COTS Component Selection. In : International Conference on Component-Based Software Systems (ICCBSS), vol. LNCS 3412 (2005)
23. Causevic, A., Krasteva, I., Land, R., Sajeev, A., Sundmark, D.: An Industrial Survey on Software Process Practices, Preferences and Methods. (2009)
24. Land, R., Alvaro, A., Crnkovic, I.: Towards Efficient Software Component Evaluation: An Examination of Component Selection and Certification. In : Euromicro SEAA SPPI Track, Parma, Italy (2008)