ISSN 1407-7493 COMPUTER SCIENCE DATORZINĀTNE 2009-7493

APPLIED COMPUTER SYSTEMS
LIETIŠĶĀS DATORSISTĒMAS

# FORTY YEARS OF SOFTWARE REUSE

# PROGRAMMATŪRAS ATKĀRTOTĀ LIETOŠANA PĒDĒJO ČETRDESMIT GADU LAIKĀ

**Vladimirs Kotovs**, M.sc.ing., Riga Technical University, Meza 1/3, Riga, LV 1048, Latvia,
vladimir.kotov@gmail.com

***Software reuse, reusable assets, history of software reuse , aspects of systematic software reuse***

## 1. Introduction

With the growing demand for rapid software system development, researchers are continuously looking for new software engineering methodologies and techniques in order to improve or automate development processes and make maintenance and support easier. The need in systematic approach and effective software engineering methods, which allow reusing experience to address recurring problems successfully, is obvious and extremely important.
Software reuse concept could be compared to the "holy grail" in software engineering [1], which has always been sought for but hard to achieve. Reuse is the process of creating software systems from existing software rather than building them from scratch [2], whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artefacts with the purpose of using them in its development activities [3], so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits [4].
A central concept in reuse research area is a reusable asset or artefact that could be either reusable software or software knowledge, potentially made up of many software life cycle products including: requirements and architecture definitions, analysis and design models, code, test programmes, test scenarios and reports. Besides different sizes and granularities, it is common to separate two main types of reusable assets: 1) vertical, which are specific to an application domain, and 2) horizontal, which can be reused independently of the application domain, with the main constraint related to asset compatibility with the application architecture [5].
Reusability as the property of a software asset, that indicates its probability of reuse, is often understood as one of the factors of software quality and development productivity.
First efforts to attain the main objective of reuse scenario - to make something once and to reuse it several times - rendered successful stories, which helped to formulate some basic principles of software reuse and induced interest. However, to benefit in more systematic and repeatable software reuse, additional research was needed.
This paper presents a review of software reuse with a brief retrospective analysis of main contributions and trends based on key researches. The objective of the article is to characterize current state in the field and to present the main aspects of systematic software reuse. The results can be used as a reference for future research in identified directions. The remaining paper is organized as follows: Sections 2 and 3 give a review of the origins and main historical contributions in the field of software reuse as Fig.1 outlines. Section 4 discusses current state, motivation, main benefits and obstacles of software reuse. Section 5 highlights main aspects of systematic application of reuse. Finally, the conclusions are made and future work is outlined in the last section.

## 2. Origins and evolution of software reuse

The notion of software reuse is as old as the software engineering field itself. From the beginning, the software reuse was positioned as a means of overcoming software crisis – the problem of building large, reliable software systems in a controlled, cost-effective way [2]. About forty years ago, during the first NATO Software Engineering Conference in 1968 McIlroy's "Mass Produced Software Components" became the seminal paper and starting point for the investigation of mass-production techniques in software engineering, concluding that "software industry is weakly founded and one aspect of this weakness is the absence of a software component sub-industry" [6].

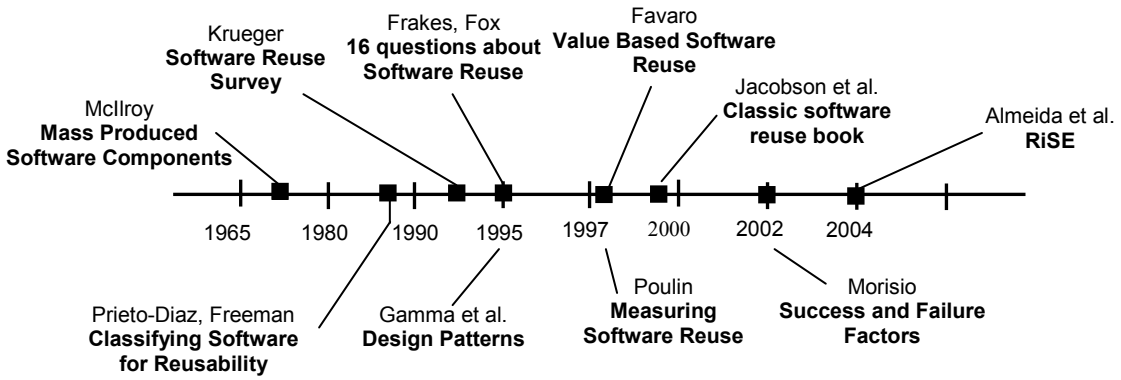


**Fig.1.** Research timeline

By analogy with hardware engineering disciplines, where reuse of standard parts is a common practice, during the early stages of research much effort was invested in the investigation of reuse libraries, e.g. classification of components, navigation in such repositories, and composition of new software out of predefined components [7]. A body of research was presented in [8], which became influential in the component retrieval area. In particular, it proposed the solution with a faceted classification scheme based on reusability-related attributes. Additionally, the authors defined a selection mechanism, that:

- allows multiple keywords to be associated to a single facet, reducing the chances of ambiguity and duplication
- proposes a component description format based on standard vocabulary of terms
- imposes citation order for the facets.

Based on observation of commonalities involved in software libraries, application generators, source code compilers, and generic software templates, the taxonomy for characterisation and comparing different approaches to software reuse was presented in [2], which characterizes each approach in terms of abstracting, selecting, specializing, and integrating its reusable artefacts.

As a result, the notion of cognitive distance was introduced as an intuitive gauge to compare the effectiveness of reuse techniques, which is informally defined as the amount of intellectual effort that must be expended by software developers in order to take a software system from one stage of development to another [2]. The research concluded that effective reuse technique must reduce the cognitive distance between the initial concept of a system and its final executable implementation, and of the eight reuse technologies examined; the reusable software architectures evaluated come closest to this description.

## 3. Evolving of systematic software reuse

The end of the 20$^{th}$ century can be characterized by evolving of systematic software reuse as a field of research in software engineering, what could be proved by the multitude of published books and broad surveys, along with various conferences and journal papers on the topic. Important contributions include analysis of organizational experience in [9], in which empirical data collected from the

responses from 29 organizations is used to answer sixteen questions (see Table 1) commonly asked by organizations attempting to implement systematic reuse.

**Table 1**

Sixteen questions about software reuse

| Question | Answer |
|---|---|
| How widely are common assets reused? | Varies |
| Do programming languages affect reuse? | No |
| Do CASE tools promote reuse? | Not yet |
| Do developers prefer to build from scratch or to reuse? | Prefer to reuse |
| Does perceived economic feasibility influence reuse? | Yes |
| Does reuse education influence reuse? | Yes |
| Does software engineering experience influence reuse? | No |
| Do recognition rewards increase reuse? | No |
| Does a common software process promote reuse? | Probably |
| Do legal problems inhibit reuse? | No |
| Does having a reuse repository improve code reuse? | No |
| Is reuse more common in certain industries? | Yes |
| Are company, division, or project sizes predictive of organizational reuse? | No |
| Are quality concerns inhibiting reuse? | No |
| Do organizations measure reuse, quality, and productivity? | Mostly not |
| Does reuse measurement influence reuse? | No |

The study highlighted some contradictions with the existed assumptions about reuse and identified the following dimensions the organizations should concentrate on to improve systematic reuse:

- education about reuse
- developers' understanding of the economic feasibility of reuse
- instituting a common development process
- making high quality assets available to developers.

An important shift from practicing reuse mostly on code level to design reuse occurred with [10], where the term of design patterns was introduced. Design patterns are known as general repeatable solutions described in a semi-formal form to commonly occurring problems in software development and helpful in understanding, modelling and implementing object-oriented micro-architectures. They capture reusable design knowledge, experience and best practices leading to extensible and reusable object-oriented designs. Reuse of design patterns helps to prevent subtle issues that can cause major problems and to avoid complexity. Among the mentioned benefits they provide the upper-level view of a problem, separation of specific details, variable and stable concerns, as well as a common vocabulary and documentation instrument among software developer community.

Other important contributions concerning the measuring of economic benefits of the software reuse and development costs were given by [11] and [12], where operational and strategic benefits were defined and quantified within the context of broader business strategy. Research concluded with a Value Based Reuse Investment (VBRI) framework for applying the disciplines of business strategy to investments in software reuse, based on the following main principles [12]:

1) Economic value maximization drives reuse investment strategies for the business
2) Strategy drives selection of reuse investments
3) Investments are actively structured to maximize embedded strategic options.

In [13] some of the factors in adopting a company-wide software reuse program were derived from empirical evidence of reuse practices from a survey of projects for the introduction of reuse in European companies. The study concluded that despite the potential for success, about one-third part of the projects failed due to

- not introducing reuse processes
- not modifying non-reuse processes

- not considering human factors
- lack of commitment by top management [13].

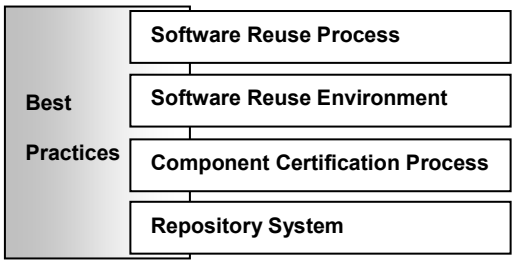


**Fig.2.** RiSE Framework components

In this context, the Reuse in Software Engineering (RiSE) project was initiated in 2004 with a goal to develop a robust framework for organizations that are moving towards an effective reuse program. Figure 2 presents an overall structure of RiSE framework adopted from [14]. The framework is composed of best practices related to software reuse, and the technical aspects of software reuse, represented by processes, environments and tools [14].

## 4. The current state in the field of software reuse

In software engineering literature different rates about the reuse could be found. Some studies have shown that 40% to 60% of code are reusable from one application to another, 60% of design and code are reusable in business applications, and 75% of functions are common for more than one application [4]. As it is demonstrated in Fig.3, theoretical reuse potential up to 85% of a new application can be achieved by reusing existing software, about 65% being domain specific and about 20% being domain independent [11].

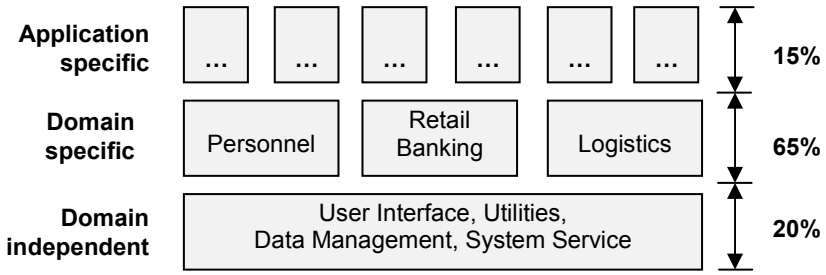


**Fig.3.** The composition of a "typical" application

However, software reuse will only succeed, if it makes good business sense, and reuse will only be chosen, if a good case can be made that it is the best alternative choice for the use of capital [15].
Table 2 consolidates the facts discussed in [16] about main benefits, that can be achieved by the organizations with the maximization of the reuse level. Table 3 indicates main obstacles that cause reuse to be not as widely practised as one might assume.

**Table 2**

Main benefits of reuse

| Benefits of reuse | Description |
|---|---|
| Gains in quality | Quality of particular component could be improved because of error corrections accumulated from reuse |
| Gains in reliability | The use of a component in several systems increases the chance of errors to be detected and improves confidence in that component |
| Gains in productivity | Productivity could be achieved due to less design and code to be developed and less testing efforts |
| Gains in performance | Extensive reuse can be worth the effort invested in optimizations, that may yield better performance of a reused component |
| Reduction of maintenance costs | Fewer defects and reduction of maintenance costs can be expected when proven quality components have been used |
| Reduction of product time to market | By using reusable artefacts organizations could reduce the product time to market which influences the success or failure of a software product |
| Rapid prototyping support | Reusable components can provide an effective basis for quick building of a prototype of a software system |

**Table 3**

Main obstacles of reuse

| Obstacles of reuse | Description |
|---|---|
| Managerial and organizational obstacles | Software reuse cannot be widely achieved in an organization without support of top-level management. Lack of management incentives prohibits managers from letting their developers spend time in making components of a system reusable |
| Economic Obstacles | Reuse requires up-front investments in infrastructure, methodology, training, tools and archives, without payoffs being realized immediately |
| Conceptual and Technical Obstacles | Concerns difficulty of finding reusable software, non-reusability of legacy components, and modification effects on the component and its previous verification results |

The following important conclusions emerge from historical experience and characterize the current state of software reuse field:

- Software reuse is multidisciplinary and has deep and complex interactions with many areas of computer science.
- The final goal is to ingrain reuse into an organization's entire software production process [18], however, each organization must analyse its own needs, define the key benefits it expects, identify and remove impediments, and manage risks [5].
- Reusing of large-scale system components (reuse-in-the-large) is a problem too hard to be solved in general way. It is more effective when systematically applied in the context of families of related systems and thus is domain-dependent.

## 5. Main aspects of systematic software reuse

Current and future research initiatives should be organized considering the three broad aspects of software reuse: organizational, economic and technical aspects.

1) At the organizational level, two sets of measures are proposed in [3] for the smooth operation of software reuse initiative:

- managerial infrastructure in the form of a set of functions, responsibilities, reporting requirements, and reward, which are required to ensure operation of reuse processes, and

- technological infrastructure that includes a configuration management and quality assurance functions to support reuse operations, as well as enforcement of testing, verification and asset certification standards.

Two types of organizational approaches are commonly observed to establishing a reuse program [15]:

- Centralized, with an organizational unit dedicated to developing, distributing, maintaining, and often providing training on reusable assets. Yet large upfront capital investment is needed for this approach; the advantages include efficient utilization of development knowledge and corporate expertise across projects and systematic management of assets.
- Distributed asset development, where reuse is implemented collaboratively by projects in the same product line. However, despite the advantages of less overhead cost needed, it may be difficult to coordinate asset development responsibilities and needs between projects.

2) From the economic perspective implementing a reuse initiative in a corporate environment requires a decision about when and where capital investment for software reuse is to be made, and whether it will be proactive or reactive [15]. Proactive approach requires a large upfront investment, and returns can be seen only when products are developed and maintained. Reactive investment is an incremental approach to asset building, when one develops reusable assets as reuse opportunities arise while developing products [15]. Economic analysis enables managers to quantify, justify, and document their decisions, thereby providing a sound basis for decision making processes. Economic aspects of software reuse are divided into three broad classes by [3]:

- metrics, which reflect attributes that increase the market value of an asset
- reuse cost estimation techniques, that derive costs and benefits for software development for reuse, and with reuse
- return-on-investment models to quantify reuse related decisions.

3) Technical aspects of software reuse can be divided into the following classes: domain engineering, component engineering and application engineering.

As it was mentioned above, the key concept of systematic reuse is the domain, which may be defined as an application area or, more formally, a set of systems that share design decisions [5]. Domain engineering activities primarily consist of scoping the domain, deriving the domain-wide product architecture, specifying the reusable assets and preparing the application development activities [3]. Existing approaches reviewed in [15] and [5] show that most of them are complementary development techniques, that focus on 1) extracting information from existing code and documents (e.g. Domain Analysis and Reuse Environment, DARE); 2) the commonality and variability analysis of the features in a product line (e.g. Feature-Oriented Reuse Method, FORM); 3) components and mechanism for integrating components (e.g. Koala); 4) modelling techniques and notations for product line engineering (Product Line UML-Based Software Engineering, PLUS). Only some approaches (e.g. FAST, RiDE) provide patterns of domain driven engineering processes with the defined steps of domain analysis, domain design, and domain implementation.

Component engineering discipline in scope of software reuse deals with the development of reusable assets. It is often treated as the most popular approach of reusing software, considering component as a software element that conforms to a component model and can be independently deployed and composed without modification according to a component standard [17]. Interest in component-based software engineering has resulted in a broad range of technologies that promote component development, integration and deployment, and provide important middleware platforms on which reusable components can be developed and integrated into applications. The most popular and mature are Object Management Group's CORBA (Common Object Request Broker Architecture), Sun's EJB (Enterprise JavaBeans), and Microsoft's COM+ (Component Object Model).

## 6. Conclusions

This work provides a brief review of software reuse based on analysis of the number of existing researches. Reuse as the process of using existing software artefacts and knowledge with more than a 40-year history is recognized as an important mechanism to improve software quality and development productivity. When properly understood and systematically deployed, reuse offers the opportunity to achieve improvements in the existing methods of software production. The paper

presents retrospective analysis of the origins and main historical contributions in the research area, which has demonstrated multidisciplinary nature of reuse and the need to ingrain reuse into the development process in order to achieve its benefits systematically. Organizational, technical and economic aspects of software reuse were reviewed briefly, highlighting the importance of the domain engineering field.

Though significant progress has already been achieved on software reuse, many important problems remain. It was noted that in order to comprehend the software reuse scenario, we should also consider conceptual reuse more actively. Another important issue concerns maturity of reuse processes without gaps in reuse activities covering steps of domain engineering, domain analysis, design and implementation. Some results in scope of the first mentioned issue have been already achieved in [19].

The future work will be focused on collecting more information and further analysis of the reuse processes. More attention is to be paid to systematic software reuse opportunities in the context of Model-Driven Development (MDD), in particular to the processes of identification of domain parts reasonable for automation.

**References**


1. Llorens J., Fuentes J.M., Prieto-Diaz R., Astudillo H. Incremental Software Reuse // Reuse of Off-the-Shelf Components. – Berlin: Springer, 2006. – P.386-389.
2. Krueger C. W., Software Reuse // ACM Computing Surveys, Vol. 24, No. 02. – New York: ACM, 1992. – P.131-183.
3. Mili H., Mili A., Yacoub S., Addy E., Reuse-based Software Engineering: Techniques, Organization, and Measurement. – John Wiley & Sons, Inc., 2002. – P.672
4. Ezran M., Morisio M., Tully C., Practical Software Reuse. – Springer, 2002. – P. 374.
5. Almeida E.S. RiDE: The RiSE Process for Domain Engineering. – 2007.
6. McIlroy M. D., Mass Produced Software Components // NATO Software Engineering Conference Report. – Garmisch, 1968. – P. 79-85.
7. Gall H., Jazayeri M., Klosch R. Research directions in software reuse: where to go from here? // ACM SIGSOFT Software Engineering Notes, Volume 20. – New York: ACM, 1995. – P.225-228.
8. Prieto-Diaz R., Freeman P., Classifying Software for Reusability // IEEE Software, Vol. 04, No. 01, - 1987. - P.6-16.
9. Frakes W. B., Fox C. J., Sixteen Questions about Software Reuse // Communications of the ACM, Vol. 38, No. 06. – New York:ACM, 1995 - P.75-87.
10. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns. Elements Of Reusable Object Oriented Software. – Addison-Wesley, 1995.
11. Poulin J. S., Measuring Software Reuse. – Addison-Wesley, 1997. – P.195.
12. Favaro J., Favaro K., and Favaro P., Value Based Software Reuse Investment // Annals of Software Engineering. vol. 5, 1998. – P.5-52.
13. Morisio M., Ezran M., Tully C., Success and Failure Factors in Software Reuse // IEEE Transactions on Software Engineering, Vol. 28, No. 04. – 2002. – P.340-357.
14. RISE Project: Towards a Robust Framework for Software Reuse / Almeida E.S., Alvaro A., Lucredio D., Garcia V.C., Meira S.R. // Information Reuse and Integration. IRI 2004. Proceedings of the 2004 IEEE International Conference. – 2004, – P. 48-53.
15. Frakes W. B., Kang K. C., Software Reuse Research: Status and Future // IEEE Transactions on Software Engineering, Vol. 31, No. 07. – 2005. – P.529-536.
16. Sametinger J., Software Engineering with Reusable Components. –Springer-Verlag, 1997.– P.275.
17. Heineman G. T., Councill W. T., Component-Based Software Engineering. – Addison-Wesley, 2001. – P.818.
18. Software Reuse and Domain Engineering / Frakes B., 2007. - http://wfrakes.wordpress.com/category/introduction/ - 02/10/2008.
19. Kotov V. Modularity improvement of design patters // ICTE in Regional Development. – Valmiera: Vidzeme University College, 2007. – P.43.-50.

***Kotovs V. Programmatūras atkārtotā lietošana pēdējo četrdesmit gadu laikā***

*Rakstā ir sniegts pārskats par programmatūras atkārtoto lietošanu, to izcelsmi, pētījumu virzieniem un svarīgākiem sasniegumiem. Atkārtotai lietošanai kā procesam, kurā ietvaros tiek izmantoti eksistējošie programmatūras artefakti un zināšanas, ir vairāk par 40 gadiem. Pašlaik tā ir atzīta par svarīgu mehānismu programmatūras kvalitātes un izstrādes produktivitātes uzlabošanai. Galvenā uzmanība ir pievērsta retrospektīvai analīzei par svarīgākiem pētījumiem programmatūras atkārtotās lietošanas jomā. Sākot no pamata pētījuma, rakstā tiek apspriesti svarīgākie pagrieziena punkti attīstībā no sākotnējām idejām par komponentu apakš-industriju. Starp svarīgākiem pētījumiem, kuriem tiek pievērsta uzmanība rakstā, ir pētījumi par atkārtotās lietošanas bibliotēkām, atkārtoti lietojamo artefaktu atlases un klasifikācijas metodēm, mērījumiem un eksperimentiem, projektēšanas šabloniem un pētījumiem par sistemātisko atkārtotu lietošanu. Speciālā uzmanība ir pievērsta programmatūras atkārtotās lietošanas labumu un kavēkļu analīzei. Izmantojot iegūto vēsturisko pieredzi, rakstā tiek secināts par atkārtotās lietošanas daudznozaru dabu, par nepieciešamību atkārtotās lietošanas procesos, un par domēnu inženierijas lomu. Ir sniegts pārskats par vairākiem aspektiem (organizācijas, tehniskiem un ekonomiskiem), kuri ir svarīgi atkārtotās lietošanas programmas īstenošanai.*

***Kotov V. Forty years of software reuse***

*This paper is an overview of software reuse, its origins, research areas and main historical contributions. Reuse as the process of using existing software artefacts and knowledge has more than 40-year long history, and is currently recognized as an important mechanism to improve software quality and development productivity. Main attention is paid to retrospective analysis of key researches in the area of software reuse. Starting from the seminal paper and the other earliest contributions the survey discusses important milestones in the evolution of initial ideas of component sub-industry to mature field of research in software engineering. Active areas of past researches being overviewed by this paper include reuse libraries, asset classification and selection, measurement and experimentation, design patterns and studies of systematic reuse. Separate attention is paid to consolidation of main benefits and obstacles of software reuse. The paper concludes important ideas emerging from the historical experience about multidisciplinary nature of reuse, necessity of software reuse process and the role of domain engineering. Overview of key aspects (organizational, technical and economic) important for establishing software reuse programs is given.*

***Котов В. Сорок лет повторного использования программного обеспечения***

*Статья является обзором повторного использования программного обеспечения, его истоков, направления исследований и основных достижений. Повторное использование программного обеспечения имеет более чем 40-летнюю историю как процесс использования существующих программных компонентов и знаний, и в настоящее время признана важным механизмом по улучшению качества программного обеспечения и скорости разработки. Главное внимание уделено ретроспективному анализу основных исследований в данной области. В статье обсуждаются важные этапы развития от первоначальных идей об индустрии программных компонентов до формирования отдельной области исследований в области разработки программного обеспечения. Основные направления прошлых исследований, проанализированных в рамках данной статьи, включают в себя библиотеки повторного использования, методы классификации и отбора компонентов, оценку и эксперимент, шаблоны проектирования и исследования систематического повторного использования программного обеспечения. Отдельное внимание уделено обобщению основных преимуществ и препятствий для реализации повторного использования. Основываясь на историческом опыте в статье делается вывод о важности идей междисциплинарного характера повторного использования, необходимости в процессе повторного использования и роли доменной инженерии. Также представлен обзор ключевых аспектов (организационных, технических и экономических), важных для реализации программ повторного использования.*