A Collaborative Method for Reuse Potential Assessment in Reengineering-Based Product Line Adoption

Muhammad Asim Noor[1], Paul Grünbacher[2], and Christopher Hoyer[1]

[1] Institute for Systems Engineering & Automation,
Johannes Kepler University, 4040 Linz, Austria
{man,hoc}@sea.uni-linz.ac.at
[2] Christian Doppler Laboratory for Automated Software Engineering,
Johannes Kepler University, 4040 Linz, Austria
paul.gruenbacher@jku.at

Abstract. Software product lines are rarely developed from scratch. Instead the development of a product line by reengineering existing systems is a more common scenario, which relies on the collaboration of diverse stakeholders to lay its foundations. The paper describes a collaborative scoping approach for organizations migrating existing products to a product line. The approach uses established practices from the field of reengineering and architectural recovery and synthesizes them in a collaborative process. The proposed approach employs best practices and tools from the area of collaboration engineering to achieve effective collaboration. The paper presents a case study as initial validation of the proposed approach.

Keywords: Reuse potential assessment, collaboration, product line adoption, product line planning.

1 Introduction

Organizations usually do not start software product lines from scratch. It is more common that organizations with successful products in a particular domain find the need to adopt a product line (PL) approach to capitalize on systematic reuse of the common functionality among existing products [1, 2]. Existing systems are the result of large investment and can not be easily discarded as they embody substantial domain knowledge and expertise. The reuse of existing assets is critical in PL adoption as developing existing systems anew for a PL is typically expensive and risky [3]. Careful planning is thus needed for the success of product line adoption. It is essential to assess the suitability of existing assets for reuse in a product line and to estimate the effort required to tailor those assets. Software products are planned, designed, and developed collaboratively by diverse people. The knowledge essential to assess the reuse potential of existing products is distributed among architects, product managers, developers, or maintainers and spread in documents and application source code [4]. Numerous formal approaches for reuse potential assessment exist in areas such as software maintenance [5, 6] or software reengineering [7-10]. These methods focus on formally captured information in documentation, models, and source code. The

© IFIP International Federation for Information Processing 2008

collaborative aspects of reuse potential assessment have so far received only little attention. Although existing approaches related to reuse potential assessment [8, 11-13] are collaborative in nature they are rather vague with respect to how effective collaboration can be achieved.

In the paper we present a collaborative and stakeholder-centric approach to reuse potential assessment. It uses stakeholders' knowledge and experience of existing systems as primary sources of information to identify the existing components and to prioritize them according to their potential for reuse. The approach also enables the team to produce working estimates of the effort needed to modify those components. Such an approach is invaluable at PL scoping and planning stage. It is conducted at a high level of granularity (i.e., logical components, subsystem or packages) to avoid getting lost in technical details.

Boehm has argued that collaborative methods are key elements of future software engineering methods [14]. We thus believe that the collaborative approach nicely complements more formal approaches. It uses proven techniques and guideline from the discipline of collaboration engineering (CE) to achieve effective collaboration. CE is an approach to designing work practices for high-value collaborative tasks [15, 16]. In CE proven patterns of group collaboration, called thinkLets, are used to describe collaborative processes [15] and to foster the interaction of individuals and teams. ThinkLets describe collaborative techniques in a compact form and can be flexibly combined to achieve the desired results. For instance, there are thinkLets that concisely describe different brainstorming and prioritization techniques. It has been shown that different software engineering tasks can be supported by composing collaborative activities from thinkLets [17, 18]. While thinkLets might appear process-centric and tool-centric at first sight they should rather be seen as facilitation techniques that are optimized to structure high-value group tasks.

The remainder of the paper is structured as follow: In Section 2 we discuss related work in the field of product line adoption, software assessment for reengineering, and collaboration engineering. Section 3 presents layers 1 and 2 of our approach and explains how the approach is supported by thinkLets. In Section 4 we present a case study and discuss its results. A conclusion and an outlook to further research round out the paper.

2 Related Work

Many methods and techniques are reported in literature [5-7, 9, 10] for assessing existing software for maintenance and evolution. These methods aim at evaluating existing software with respect to business value and technical value to identify promising candidates for reengineering (e.g. [5]). The technical value is determined by variables such as maintainability, decomposability, deterioration, or obsolescence. The Product Line Practice Framework by Clements *et al.* [1] represents a comprehensive framework dealing with all aspects of product lines from development to evolution. It provides high level guidance for mining existing assets. The approach makes use of the options analysis for reengineering (OAR) method [8] and the mining architecture for product line (MAP) method [12] for identifying existing assets to be reused in product line development. However, the framework does not shed light on the collaborative aspects of this process [8, 12].

Bergey *et al.* [8] define software reengineering as "transforming an existing design of a software system (or element of that systems) to a new design while preserving the system's intrinsic functionality". Traditional reengineering approaches start with the analysis of legacy assets, the extraction of design and architectural information followed by an exploration of the options and possibilities, and the implementation of the best option (e.g. [13]). The *Horse shoe* model presented as part of *OAR* in [8] is an example of such an approach. SRAH [7] is a process for assessing legacy software to select the best options for legacy software evolution and to ease maintenance. The output of the process is a succinct report on which senior management can make informed decisions. Kolb *et al.* report on a case study [19] about the use of refactoring techniques to evolve and adapt existing components for reuse in a product line.

Several authors have addressed product line scoping and planning. For instance, Schmid [20] proposes a three staged approach comprised of product mapping, domain potential analysis and reuse infrastructure scoping. We presented a collaborative product mapping approach in the context of product line adoption in earlier work [4, 21, 22] based on Schmid's framework. In [23] Schmid explores the economic impact of product line adoption and evolution and identifies the four adoption strategies: big bang, project integration, reengineering-based, and leveraged (deriving a product line from another product line). In reengineering-based product line adoption the scoping activities gain a different focus as the product map guides the extraction of features from legacy systems as suggested in [4, 21, 22]. Other work on product line adoption can be found in a case study by Bayer *et al.* [24] who report on a migration process guided by the RE-PLACE approach. In [25] Ebert *et al.* identify a clear business focus, strong release planning, and requirements management as success criteria for product line adoption. Kircher *et al.* in [26] discuss challenges in product line adoption and report a set of best practices.

The discipline of collaboration engineering provides a wide range of practices, patterns and tools to achieve effective collaboration. Collaboration engineering aims at designing work practices for practitioners to support high-value recurring collaborative tasks [15]. There are six general patterns of collaboration: generate, reduce, clarify, organize, evaluate, and build consensus [27]. The approach tries to provide the efficiency and effectiveness of professional facilitators to the practitioners who are not experts in team interaction. ThinkLets [16] describe patterns for collaborative activities and have become widely accepted building blocks for designing collaborative processes. A thinkLet is a named, scripted, and well-tested activity that produces a known pattern of collaboration among people working together on a common goal [28]. There are currently about 70 well-documented thinkLets [29] some of which are used in our approach to reuse potential assessment. Many collaborative processes have been successfully designed using thinkLets. An example is the requirements negotiation method EasyWinWin which incorporates a number of agile principles [30]. Our earlier work [4, 21, 22] also suggest that collaborative techniques are valuable and useful in product line planning.

3 A Collaborative Process for Reuse Potential Assessment

Fig. 1 shows the involved participants, inputs and outputs of the Reuse Potential Assessment (RPA) process. The process relies on the knowledge and experience of the participants, available documentation and analyses of the systems to be reused, and the proposed product map of the future product line.

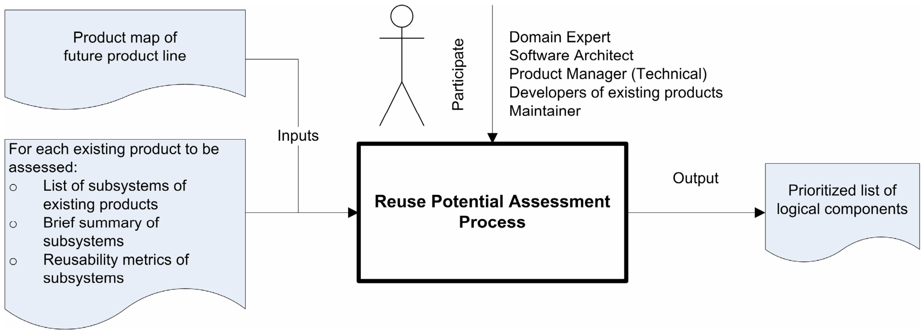

Product map of future product line
For each existing product to be assessed:
List of subsystems of existing products
Brief summary of subsystems
Reusability metrics of subsystems
Inputs
Participate
Domain Expert
Software Architect
Product Manager (Technical)
Developers of existing products
Maintainer
Reuse Potential Assessment Process
Output
Prioritized list of logical components

Fig. 1. Participants, inputs and output of the reuse potential assessment process

The selection of the right participants is a key factor for the success of the collaborative RPA process [31]. The selection must be based upon the knowledge, experience and expertise of people with the products to be assessed. The inclusion of domain experts and software architects in the team is essential. The number of technical experts needed for the RPA process for a particular product depends on the size and complexity of the products to be assessed.

A product map as defined in [4, 21, 22] is an important input to the RPA process. It is used to ensure a shared understanding about the common functionality among the products of the PL and helps the team to identify logical components from the existing systems. A further input to the RPA process is a list of subsystem for each of the products to be assessed. A brief summary explaining the functionality of the subsystems is also provided. Furthermore, reusability metrics for the subsystems are extracted beforehand and are provided as part of the subsystem summary. Similar to existing models for reuse potential assessment [5, 8, 10] our RPA process makes use of static analyses of the existing systems. The metrics used to evaluate the reusability are size (e.g., file size method size), complexity (e.g., cyclomatic complexity, boolean expression complexity, nesting levels), decomposability (e.g., n tier architecture), dependencies (e.g., data abstraction coupling, fan-out), or understandability (e.g., ratio of non commented line of code, naming) [5]. The static analysis can be facilitated by tools, e.g., Checkstyle[1]. Commercial IDEs (e.g. IntelliJIDEA) support static analysis on the desired levels of abstraction (e.g., method, class, or package).

We describe the collaborative RPA process on three layers of abstraction: Fig. 2 shows the highest layer 1, i.e., the tasks of the process, input and output of the tasks, the collaboration patterns used in the execution of the task, and the sequence of the

[1] http://eclipse-cs.sourceforge.net/index.shtml

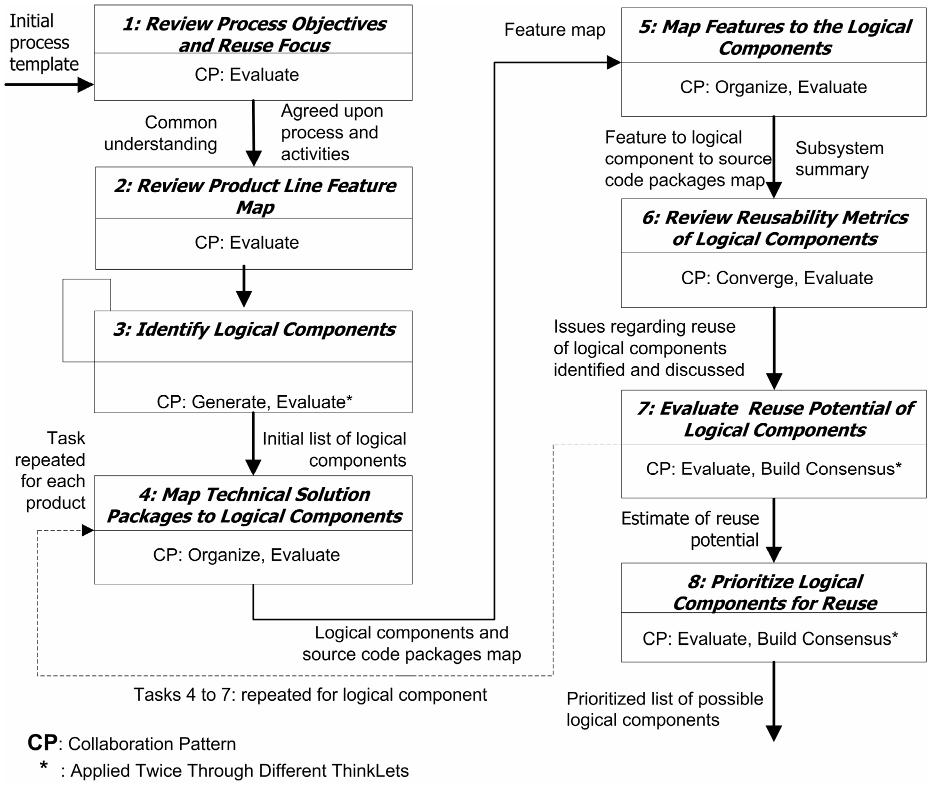

Initial process template
1: Review Process Objectives and Reuse Focus
CP: Evaluate
Common understanding
Agreed upon process and activities
2: Review Product Line Feature Map
CP: Evaluate
3: Identify Logical Components
CP: Generate, Evaluate*
Initial list of logical components
Task repeated for each product
4: Map Technical Solution Packages to Logical Components
CP: Organize, Evaluate
Logical components and source code packages map
Tasks 4 to 7: repeated for logical component
Feature map
5: Map Features to the Logical Components
CP: Organize, Evaluate
Feature to logical component to source code packages map
Subsystem summary
6: Review Reusability Metrics of Logical Components
CP: Converge, Evaluate
Issues regarding reuse of logical components identified and discussed
7: Evaluate Reuse Potential of Logical Components
CP: Evaluate, Build Consensus*
Estimate of reuse potential
8: Prioritize Logical Components for Reuse
CP: Evaluate, Build Consensus*
Prioritized list of possible logical components
CP: Collaboration Pattern
* : Applied Twice Through Different ThinkLets

Fig. 2. Task view of the RPA process (Layer 1)

execution of the tasks. At layer 2, we show how the tasks and collaboration patterns are supported by thinkLets (cf. Fig. 3). Layer 3 (cf. case study section 4) describes a concrete enactment of the process during a pilot case study demonstrating the actual use of collaborative tools.

Fig. 2 shows the process tasks and their associated thinkLets. There are seven thinkLets used in the process: The thinkLet *ReviewReflect* facilitates a group to review an outline or a document. Team members collaboratively go through the outline and record their thoughts and suggestion by adding comments. Right after, these ideas are discussed in a moderated fashion. Consolidated recommendations are prepared or changes to the documents are made. The thinkLet *BucketWalk* aims at achieving a shared vision amongst groups of people by a collaborative walkthrough of all the items in different categories while encouraging the discussion for issues and demanding explanation. The group does not move forward before open issues are resolved. The thinkLet *LeafHopper* aims at eliciting ideas from participants regarding a set of topics. The thinkLet *PopcornSort* helps structuring collected raw ideas into appropriate categories.

The thinkLet *StrawPoll* enables decision-making through measuring opinions of the participants in quantitative terms. A wide variety of voting methods can be used

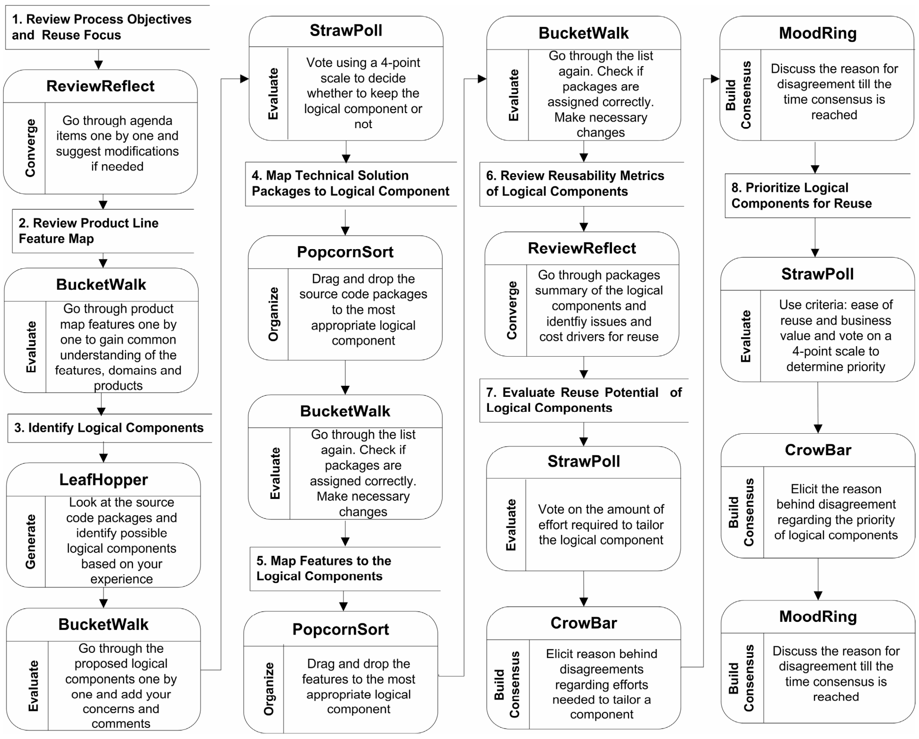

1. Review Process Objectives and Reuse Focus
ReviewReflect
Converge
Go through agenda items one by one and suggest modifications if needed
2. Review Product Line Feature Map
BucketWalk
Evaluate
Go through product map features one by one to gain common understanding of the features, domains and products
3. Identify Logical Components
LeafHopper
Generate
Look at the source code packages and identify possible logical components based on your experience
BucketWalk
Evaluate
Go through the proposed logical components one by one and add your concerns and comments
StrawPoll
Evaluate
Vote using a 4-point scale to decide whether to keep the logical component or not
4. Map Technical Solution Packages to Logical Component
PopcornSort
Organize
Drag and drop the source code packages to the most appropriate logical component
BucketWalk
Evaluate
Go through the list again. Check if packages are assigned correctly. Make necessary changes
5. Map Features to the Logical Components
PopcornSort
Organize
Drag and drop the features to the most appropriate logical component
BucketWalk
Evaluate
Go through the list again. Check if packages are assigned correctly. Make necessary changes
6. Review Reusability Metrics of Logical Components
ReviewReflect
Converge
Go through packages summary of the logical components and identfiy issues and cost drivers for reuse
7. Evaluate Reuse Potential of Logical Components
StrawPoll
Evaluate
Vote on the amount of effort required to tailor the logical component
CrowBar
Build Consensus
Elicit reason behind disagreements regarding efforts needed to tailor a component
MoodRing
Build Consensus
Discuss the reason for disagreement till the time consensus is reached
8. Prioritize Logical Components for Reuse
StrawPoll
Evaluate
Use criteria: ease of reuse and business value and vote on a 4-point scale to determine priority
CrowBar
Build Consensus
Elicit the reason behind disagreement regarding the priority of logical components
MoodRing
Build Consensus
Discuss the reason for disagreement till the time consensus is reached

Fig. 3. ThinkLets view of the process (Layer II)

with this thinkLet. The thinkLet *CrowBar* helps to elicit reasons for discord. It is usually used after the thinkLet *StrawPoll*, which highlights the agreements and disagreements resulting from certain issues. The thinkLet *MoodRing* aims at building consensus. It is usually used together with the thinkLet *CrowBar*, which highlights reasons for disagreement. These reasons are discussed in a moderated fashion. During the discussion persons originally disagreeing can change their mind and change their vote anonymously.

More specifically the purpose of each task in the collaborative process is as follows:

Task 1: Review Process Objectives and Reuse Focus. The facilitator (i) communicates to the participants the objectives of the overall process and the agenda and (ii) fine-tunes the process in light of the participants' input. The involved stakeholders include product managers, architects, developers, maintainers and domain experts. The participants collaboratively review each task of the agenda. The team agrees on the focus of reuse at this stage, i.e., the principal elements of interest for a team meeting (e.g., particular areas of the source code, algorithms, GUIcomponents, documentation, test cases or test data, etc.). This task is supported by the thinkLet *ReviewReflect.*

Task 2: Review Product Line Feature Map. The RPA process relies on a product map as input, which can be defined collaboratively as outlined in [4, 21, 22]. It is important for the team to develop a shared vision regarding the scope and vision of the product line before mining for potential assets. The participants familiarize themselves with the product map of the product line under development, which is described in terms of features, domains and products. They use the thinkLet *BucketWalk* to collaboratively navigate through the product map and to suggest changes. This helps to create a shared vision among participants regarding the scope and structure of the product line. This knowledge is essential to identify correct logical components, which may be suitable to be reused as PL core assets. Schmid [20] suggests to use domains to ease the task of identifying core assets for a PL. Domains are defined as relatively independent coherent clusters of functionality that contain one or more features. Features represent externally visible characteristics of systems (e.g., software project tracking in a product line for project management).

Task 3: Identify Logical Components. In this task participants identify logical components from existing products, which are then further investigated for their suitability to be included in the product line. A logical component can be seen as an abstract core asset of the envisaged PL. A logical component can be realized by either adapting existing implementation or by developing them anew. However, the focus of our approach is on reuse of existing assets. That is why we discuss only the case of adaptation. The challenge of this task is to identify solution elements (e.g., classes, modules, subsystems, libraries) from existing systems as candidate core assets. It is essential to balance the desire for high quality core assets and the effort required to adapt the existing technical implementation. The output of the task is a list of logical components for every existing product. The task is based on a collaborative walk-through of the modules of a product. For instance, in case of a Java-based system, modules are packages with a brief summary and the list of constituent classes. Assuming that the participants are well familiar with the products, they identify candidate logical components based on the information presented to them. The collection of the logical components is accomplished by executing the thinkLet *LeafHopper* which uses directed brainstorming. The participants brainstorm candidate components to a shared list. People see what other logical components have been suggested by other participants and they can add comments. The thinkLet *BucketWalk* facilitates common understanding and refinement of the list of logical components through moderated discussion. The thinkLet *StrawPoll* (electronic voting) may be conducted to reach consensus within the team whether a proposed logical component should be kept for further investigation or not. This task is repeated for every existing product and results into a list of logical components for every investigated system.

Task 4: Map Technical Solution Packages to Logical Components. Participants identify links between the logical components and the technical solution packages of the existing product. For instance, dependencies are established between existing modules or subsystems which implement the functionality of a logical component. The scope and definition of logical components are refined where necessary. The thinkLet *PopcornSort* facilitates the assignment of implementation units (e.g., modules or classes or packages) to the appropriate logical components and helps bounding the scope of the logical component. The thinkLet *BucketWalk* ensures consensus among participants about the appropriateness of the scope of the logical components. This task is repeated for each prospective product and results in improved definitions of the logical components. The results can be refined in task 6 when performing a more detailed analysis of the existing system. This task can be supported by feature location approaches (e.g., [32]) or scenario-based traced analysis techniques [33] depending on the complexity of the system and the knowledge of the stakeholders.

Task 5: Map Features to the Logical Components. Participants identify links between logical components and the features from the product map. This task is performed in a similar manner as the previous task using the thinkLets *PopcornSort* and *BucketWalk.* Each feature is assessed and assigned to the appropriate logical component. The task is repeated for every product. The goal of tasks 4 and 5 is to establish initial coarse-grain traceability between features, logical components and source code. This traceability allows the visualization of the logical components and eases later design activities. Even if traceability links exist (e.g., between requirements, design artefacts and source code) the above two tasks may be performed to take into account the new abstraction layer of logical components.

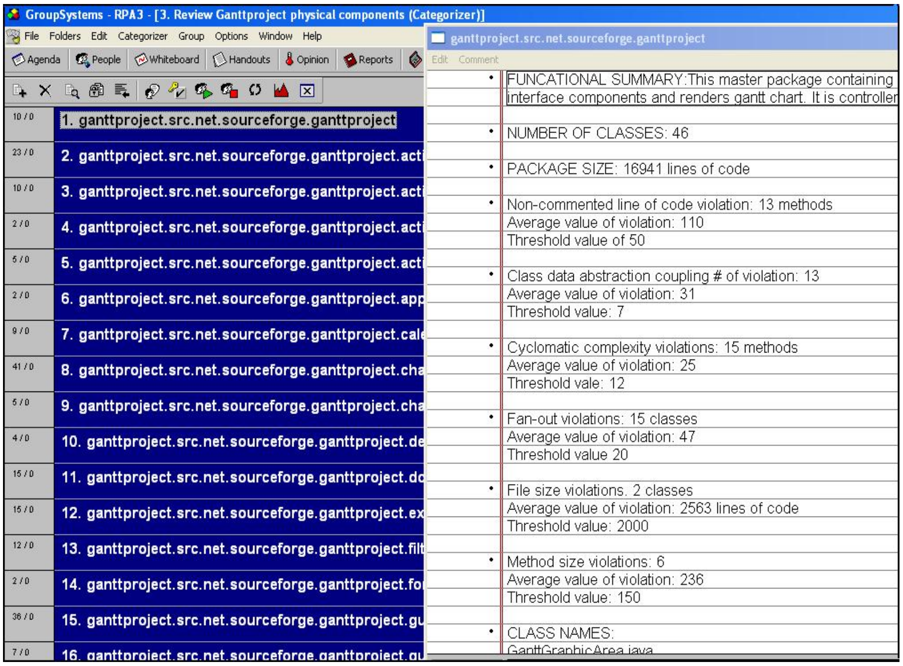
Task 6: Review Reusability Metrics of Logical Components. Participants review the logical components using the thinkLet *ReviewReflect.* For each package thought to be reusable in the logical component, they go through the information provided in the subsystem summary. Mainly, the different implications for the efforts required for reusing the package are reviewed. First, participants collaboratively go through the information (functional summary and metrics of each package) and add their opinion. Questions they try to answer include 'What are the challenges in reusing this package?' or 'What reengineering techniques are suitable for this package?'. Later, the collected comments are discussed in a moderated fashion and a consolidated list of issues and possible solution is created for each logical component.

Task 7: Evaluate the Reuse Potential of Logical Components. Participants estimate the costs and effort required to adapt the logical component as a core asset of the product line. The reuse effort estimation of the participants can be elicited through the thinkLet *StrawPoll,* where participant have to assess the level of effort required to tailor a particular logical component. In case stakeholders cannot agree about the efforts required to tailor a certain logical components the thinkLet *CrowBar* is used followed by the thinkLet *MoodRing* to reveal the reasons for disagreement. This process is repeated for each logical component. The result of this task is an initial estimate of cost/effort for adapting the logical components. These estimates are intended for selecting the most promising components for adaptation during planning while more formal cost estimation approaches can be used at design time. The task is repeated for each product.

Task 8: Prioritize Logical Components for Reuse. In this task the logical components are prioritized for further investigation at design time and later adaptation. It is accomplished through the thinkLet *StrawPoll,* which is conducted by electronicvoting. Participants assign values on a scale of 4 to each logical component. Two parameters are used: (i) the value of reuse and (ii) the effort required to reuse the logical component. Logical components which require less tailoring effort and have the highest business value will be assigned a top priority. In case of significant differences of opinion the thinkLets *CrowBar* and *MoodRing* are executed as in the previous task. This task concludes the RPA process and produces a list of the mostpromising candidate logical components for further adaptation and refinement as core assets.

4 Initial Evaluation

We conducted a pilot case study to assess the usefulness of the proposed collaborative process and the usability of the supporting tools. The study was a fictitious organisation developing a product line for project management tools based on open source code assets. In order to define the desired product line, a group of three domain experts developed a product map containing 120 features, 14 domains and three products for the project management domain. The feasibility study was based on the open source systems Gantt Project[2] and Project Factory[3]: The size of Gantt Project is 51 packages, 492 classes and is 63 KLOC. The size of Project Factory is 16 packages, 140 classes and 29 KLOC.


GroupSystems - RPA3 - [3. Review Ganttproject physical components (Categorizer)]
File Folders Edit Categorizer Group Options Window Help
Agenda People Whiteboard Handouts Opinion Reports
ganttproject.src.net.sourceforge.ganttproject
Edit Comment

10 / 0 1. ganttproject.src.net.sourceforge.ganttproject
23 / 0 2. ganttproject.src.net.sourceforge.ganttproject.acti
10 / 0 3. ganttproject.src.net.sourceforge.ganttproject.acti
2 / 0 4. ganttproject.src.net.sourceforge.ganttproject.acti
5 / 0 5. ganttproject.src.net.sourceforge.ganttproject.acti
2 / 0 6. ganttproject.src.net.sourceforge.ganttproject.app
9 / 0 7. ganttproject.src.net.sourceforge.ganttproject.cale
41 / 0 8. ganttproject.src.net.sourceforge.ganttproject.cha
5 / 0 9. ganttproject.src.net.sourceforge.ganttproject.cha
4 / 0 10. ganttproject.src.net.sourceforge.ganttproject.de
15 / 0 11. ganttproject.src.net.sourceforge.ganttproject.do
15 / 0 12. ganttproject.src.net.sourceforge.ganttproject.ex
12 / 0 13. ganttproject.src.net.sourceforge.ganttproject.filt
2 / 0 14. ganttproject.src.net.sourceforge.ganttproject.fo
36 / 0 15. ganttproject.src.net.sourceforge.ganttproject.gu
7 / 0 16. ganttproject.src.net.sourceforge.ganttproject.gu

- FUNCATIONAL SUMMARY:This master package containing interface components and renders gantt chart. It is controller
- NUMBER OF CLASSES: 46
- PACKAGE SIZE: 16941 lines of code
- Non-commented line of code violation: 13 methods
  Average value of violation: 110
  Threshold value of 50
- Class data abstraction coupling # of violation: 13
  Average value of violation: 31
  Threshold value: 7
- Cyclomatic complexity violations: 15 methods
  Average value of violation: 25
  Threshold vale: 12
- Fan-out violations: 15 classes
  Average value of violation: 47
  Threshold value 20
- File size violations. 2 classes
  Average value of violation: 2563 lines of code
  Threshold value: 2000
- Method size violations: 6
  Average value of violation: 236
  Threshold value: 150
- CLASS NAMES:
  GanttGraphicArea.java

Fig. 4. Package summary containing reusability metrics

Three engineers participated in this case study. The process was conducted in three workshops with duration of approximately 4 hours each. As preparation for the workshop the moderator (i) defined the agenda according to the tasks identified in this paper, (ii) uploaded the feature map of the product line to the collaboration tool GroupSystems, and (iii) uploaded the package list and package summaries containing the reusability metrics to the collaborative tool (see Figure 4). The feature map was developed prior to the workshop following the method described in [4, 21, 22].

[2] http://sourceforge.net/projects/ganttproject
[3] http://sourceforge.net/projects/projectfactory/

Table 1. Metrics applied in the static analysis

Metric	Level	Description	Value
Cyclomatic Complexity	Method	A measure for the minimum number of possible paths through the source and therefore the number of required tests.	10
Coupling	Class	A measure for the number of instantiations of other classes within the given class.	7
Fan-out	Class	A measure for the number of other classes a given class relies on.	20
NCSS	Method	A measure for the number of non-commented source statement within a method.	50
BEC	Line	A measure for the number of &&, || and ^ in an expression. (BEC = Boolean expression complexity).	3
File Size	Class	A measure of the file size in lines of code.	2000
Method Size	Method	A measure for the method size in lines of code.	150

Table 2. Gantt project metrics

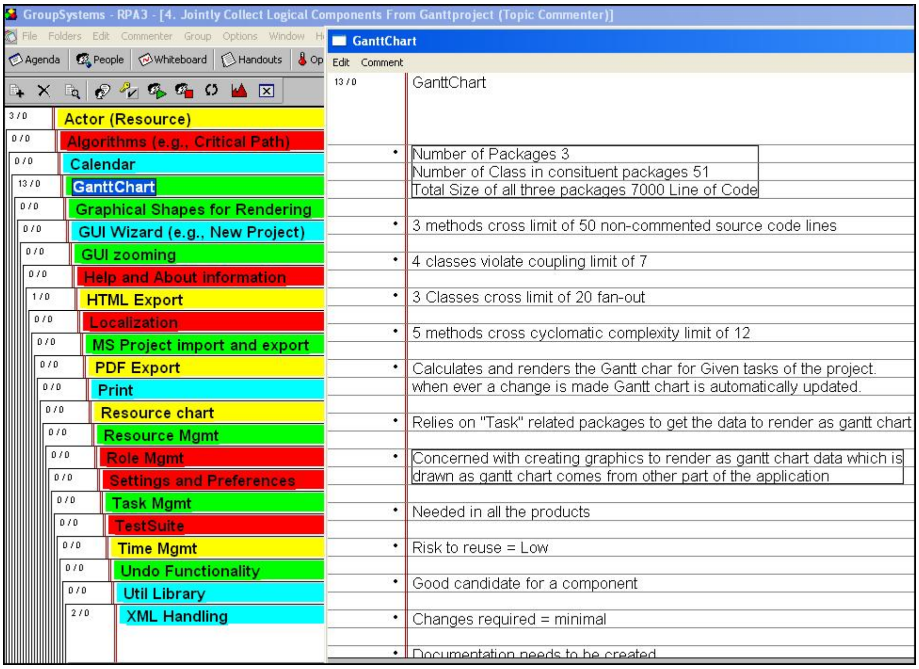
Metric	# of violations	Average violations value
Cyclomatic complexity	45	15
Coupling:	51	18
Fan-out	36	38
NCSS (Non-commented lines of code)	35	97
BEC (Boolean expression complexity)	3	9
File Size	2	2539
Method Size	12	230

In order to extract the reusability metrics a static analysis was conducted by one software engineer for both products. Source code metrics such as cyclomatic complexity, data abstraction coupling, fan-out, non commented line of code, size of the class and size of the methods, are reported in literature to be useful indicators of the reusability of the code [19, 34]. Generally, it is assumed that the lower the value of above mentioned metrics the more reusable is that source code [34]. These combined metrics were used to complement the overall picture and help to identify the reusable software elements.

Cyclomatic complexity, non-commented line of code, and size of method were measured at method level. The remainder at class level. We used the open source tool CheckStyle (available as an Eclipse plug-in) to calculate these metrics. The default threshold values of CheckStyle (for above mentioned metrics) were used. These values concur with existing literature in the field of software maintenance and evolution [19, 35]. Table 1 shows a brief description of the metrics, along with the default threshold values of CheckStyle.

Table 2 shows the consolidated summary of these metrics for one of the products assessed, i.e., Gantt project. For example, among 492 classes of Gantt Project 51 classes have a class data abstraction coupling of more than 7. Among these 51 classes the average class data abstraction coupling is 18. Further analysis shows that these are mainly those classes which primarily deal with the GUI (in the case of Gantt Project they use Java Swing components). This indicates coupling is not a big hindrance when reusing the Gantt Project source code.

Overall, these metrics indicate that the Gantt Project implementation is not overly complex as only 45 methods exceed the threshold complexity value. Mostly, these


GroupSystems - RPA3 - [4. Jointly Collect Logical Components From Ganttproject (Topic Commenter)]
File Folders Edit Commenter Group Options Window
Agenda People Whiteboard Handouts
GanttChart
Edit Comment

3 / 0 Actor (Resource)
0 / 0 Algorithms (e.g., Critical Path)
0 / 0 Calendar
13 / 0 GanttChart
0 / 0 Graphical Shapes for Rendering
0 / 0 GUI Wizard (e.g., New Project)
0 / 0 GUI zooming
0 / 0 Help and About information
1 / 0 HTML Export
0 / 0 Localization
0 / 0 MS Project import and export
0 / 0 PDF Export
0 / 0 Print
0 / 0 Resource chart
0 / 0 Resource Mgmt
0 / 0 Role Mgmt
0 / 0 Settings and Preferences
0 / 0 Task Mgmt
0 / 0 TestSuite
0 / 0 Time Mgmt
0 / 0 Undo Functionality
0 / 0 Util Library
2 / 0 XML Handling

13 / 0 GanttChart

- Number of Packages 3
  Number of Class in consituent packages 51
  Total Size of all three packages 7000 Line of Code
- 3 methods cross limit of 50 non-commented source code lines
- 4 classes violate coupling limit of 7
- 3 Classes cross limit of 20 fan-out
- 5 methods cross cyclomatic complexity limit of 12
- Calculates and renders the Gantt char for Given tasks of the project. when ever a change is made Gantt chart is automatically updated.
- Relies on "Task" related packages to get the data to render as gantt chart
- Concerned with creating graphics to render as gantt chart data which is drawn as gantt chart comes from other part of the application
- Needed in all the products
- Risk to reuse = Low
- Good candidate for a component
- Changes required = minimal
- Documentation needs to be created

Fig. 5. Consolidate list of issues of a logical component (output of task 6)

methods handle XML tags as data is stored in XML files. Most classes are within an acceptable range of coupling and fan-out. There are only two classes and 12 methods which violate the modest limit (2000, 150 LOC). The code is mostly well commented. The extracted metrics indicate that components can be extracted from Gantt Project implementation without excessive difficulty. Similar metrics were extracted from Project Factory but are not reported in this paper due to space limitations. These metrics were added in the package summary for easy perusal of the participants. The package level summary serves as a quick reference of the package implementation and is used to define the logical components (task 3) and to review the reusability metrics of the logical component (task 6).

In the following, we describe the enactment of each task in the pilot case study: The first two tasks were performed once in the beginning whereas task 3 was repeated for the two analyzed products. Tasks 4 to 7 were repeated for each logical component. The first tasks (*Review process objective and reuse focus* and *Review product line feature map*) were accomplished by conducting the thinkLets *ReviewReflect* and *BucketWalk* respectively. The small number of participants who had already jointly developed the process and product map earlier simplified these tasks. However, in more realistic settings a presentation about the agenda would be needed to explain the purpose and objectives of the exercise. The focus of the reuse was on GUI elements, algorithms and cohesive functionality (e.g., forecasting, Gantt chart) in the source code.

The third task was to identify logical components from the source code. In Fig. 5 coloured entries represent the identified logical components. Participants used the thinkLet *LeafHopper* to identify the logical components. This task was performed for both products simultaneously. In total the team identified 23 logical components from Gantt Project. Many logical components were later found to be of too limited size or use but no new components were added. Different team members had identified logical components at different level of granularity, which did not raise problems as logical components were refined in subsequent steps. Logical components were also filtered out if considered as inappropriate based on selected criteria (e.g. the minimum size of the implementation encompassed by the logical component).

Due to the tight schedule of the team members at the time of the case study task 4 (*Map Technical Solution Packages to Logical Components)* and task 5 (*Map Features to the Logical Components*) were performed asynchronously. This allowed creating a better visualization of the logical components, which was also necessary to support the detailed analysis in task 6 as detailed traceability reports where unavailable for the selected open source applications.

Task 6 (*Review reusability metrics of logical component*) was accomplished by conducting the thinkLet *ReviewReflect*. The participants created a consolidated list of issues and opinions for each logical component as shown in Fig. 5. This task aimed at collecting information about the reusability of packages (in order to realize the logical component) from the calculated metrics and the technical knowledge and experience of the people with these packages.

Task 7 (*Evaluate the Reuse Potential of Logical Components*) is supposed to be performed directly after task 6 so that participants still have the findings of the previous step in their minds. However, in our case study the team did not perform this task. Estimates of effort and cost can only be made on the basis of above mentioned information if the team has in depth knowledge of the system. Such knowledge is available only for people that have been involved in the design, development or maintenance of the product. The teams in our case did not have such an intimate knowledge. Instead, the prioritization of the logical components was performed directly after task 7.

Table 3. Prioritized high level logical components

Logical Component	# of Packages	# of Classes	Total Size
GUI Components	11	103	13 KLOC
Task Management	6	67	7 KLOC
Gantt Chart	3	51	7 KLOC
IO Handling	4	51	6 KLOC
Calendar and Time Mgmt	3	31	3 KLOC
Actor (resource) Mgmt	2	19	2 KLOC
Test Suite	1	22	2 KLOC
Project Forecast (Factory)	NA	2	1 KLOC
Project Management	1	46	17 KLOC

Lastly, the prioritization (task 8) was done using a collaborative voting tool. First, the team members assessed the business value and the ease of reuse of each component on a scale of 4. The average of these two values determined the priority of the component. Table 3 shows the list of prioritized high level logical components beginning with the components having highest priority.

These components have a similar business value as all are important in a project management application. Their priority is mainly determined on the basis of ease of reuse.

We started the case study with two products offering quite similar functionality. We aimed at identifying reusable components from these two products which can be modified and used as core assets of a product line in the project management domain. Out of 9 components identified for reuse, 8 come from Gantt Project and only the component "Project Forecast" comes from Project Factory.

It is essential that people with intimate technical knowledge of the products participate in the reuse potential assessment. Without such knowledge identifying relevant logical components and creating traceability links between features, logical component and physical component of the technical solution is difficult. The case study team also suffered to some extent from these problems due to a lack of in-depth knowledge of the products. It identified 9 components that can be reused as core assets in the product line (as shown in table 3) based on an initial list of 24 and 13 candidate components from Gantt Project and Factory respectively.

5 Conclusions and Future Work

We presented a collaborative approach for reuse potential analysis that is intended to complement more formal approaches for reengineering legacy assets in the area of product line planning. The process aims at supporting a team to collaboratively identify components with a high reuse potential from different legacy products. The process also increases the understanding of traceability and the dependencies between features and technical solution components and provides initial estimates for the effort of reuse. The presented process relies on the careful selection of stakeholders to ensure the knowledge and experience required. Absence of such knowledge and experience will undermine the collaborative aspects of the process and force the team to rely on more formal approaches, i.e., reverse engineering.

Due to our experience with collaboration engineering methods and thinkLets in other areas of software engineering such as requirements negotiation, risk management, or software inspection we expect this collaborative process to scale well in a real-world product line setting. The experience gained in the feasibility study confirms these findings. We will use the process in near future with an industrial partner specialized in ERP solutions who is currently shifting to a product line approach. The experiences also confirm that thinkLets can be effectively supported by collaborative tools, in our case a Group Support System (GSS[4]) tool was used to support the stakehoder collaboration.

References

1. Clements, P., Northrop, L.: Software product lines: practices and patterns. Addison-Wesley, Reading (2002)
2. Boeckle, G., Clements, P., McGregor, J.D., Muthig, D., Schmid, K., Siemens, A.G., Munich, G.: Calculating ROI for software product lines. IEEESoftware 21(3), 23–31 (2004)

[4] http://www.groupsystems.com

3. Aversano, L., Tortorella, M.: An assessment strategy for identifying legacy system evolution requirements in eBusiness context. J. Softw. Maint. Evol.: Res. Pract. 16, 255–276 (2004)
4. Noor, M.A., Grünbacher, P., Briggs, R.O.: Defining a Collaborative Approach for Product Line Scoping: A Case Study in Collaboration Engineering. In: IASTED Conference on Software Engineering (SE 2007) Innsbruck, Austria, February 13 (2007)
5. De Lucia, A., Fasolino, A.R., Pompelle, E.: A decisional framework for legacy system management. In: Proceedings of IEEE International Conference on Software Maintenance, pp. 642–651 (2001)
6. Ransom, J., Sommerville, I., Warren, I.: A Method for Assessing Legacy Systems for Evolution. In: Proceedings of Reengineering Forum, p. 98 (1998)
7. Software Engineering Assessment HandBook Version 3, DoD US (1997) last checked: 9-08-07, `http://www.swen.uwaterloo.ca/~kostas/ECE750-3/srah.pdf`
8. Bergey, J.K., O'Brien, L., Smith, D.: Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets 2001. Carnegie Mellon University, Software Engineering Institute
9. Caldiera, G., Basili, V.R.: Identifying and qualifying reusable software components. IEEE Computer 24(2), 61–70 (1991)
10. Sneed, H.M.: Planning the reengineering of legacy systems. IEEE Software 12(1), 24–34 (1995)
11. De Baud, J.M., Flege, O., Knauber, P.: PuLSE-DSSA—a method for the development of software reference architectures. In: Proceedings of the third international workshop on Software architecture, pp. 25–28 (1998)
12. O'Brien, L., Smith, D.: MAP and OAR Methods: Techniques for Developing Core Assets for Software Product Lines from Existing Assets. Carnegie Mellon University, Software Engineering Institute (2002)
13. Stoermer, C., O'Brien, L.: MAP-Mining Architectures for Product Line Evaluations. In: Proceedings of the IEEE/IFIP Working Conference on Software Architectures, Amsterdam, The Netherlands, August 2001, pp. 35–44 (2001)
14. Boehm, B.: A view of 20th and 21st century software engineering. In: Proceeding of the 28th international conference on Software engineering (ICSE 2006), pp. 12–29 (2006)
15. Briggs, R.O., d.V.G.J., Nunamaker, J.J.F.: Collaboration Engineering with ThinkLets to Pursue Sustained Success with Group Support Systems. Journal of Management Information Systems 19(4), 31–64 (2003)
16. Briggs, R.O., De Vreede, G.J., Nunamaker Jr, J.F., Tobey, D.: ThinkLets: achieving predictable, repeatable patterns of group interaction with group support systems (GSS). In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, p. 9 (2001)
17. Grünbacher, P., Halling, M., Biffl, S.: An empirical study on groupware support for software inspection meetings. In: 18th IEEE International Conference on Automated Software Engineering, pp. 4–11 (2003)
18. Grünbacher, P., Seyff, N., Briggs, R.O., In, H.P., Kitapci, H., Port, D.: Making every student a winner: The WinWin approach in software engineering education. Journal of Systems and Software 80(8), 1191–1200 (2007)
19. Kolb, R., Muthig, D., Patzke, T., Yamauchi, K.: Refactoring a legacy component for reuse in a software product line: a case study. Journal of Software Maintenance and Evolution: Research and Practice 18, 109–132 (2006)
20. Schmid, K.: A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering, pp. 593–603 (2002)
21. Noor, M.A., Rabiser, R., Grünbacher, P.: A Collaborative Approach for Reengineering-based Product Line Scoping in APLE - 1st International Workshop on Agile Product Line Engineering 2006, Baltimore, Maryland (2006)

22. Noor, M.A., Rabiser, R., Grünbacher, P.: Agile Product Line Planning: A Collaborative Approach and a Case Study. Journal of Systems and Software (to appear), doi:10.1016/j.jss.2007.10.028
23. Schmid, K., Verlage, M.: The Economic Impact of Product Line Adoption and Evolution. IEEE Software 19(4), 50–57 (2002)
24. Bayer, J., Girard, J.F., Wuerthner, M., De Baud, J.M., Apel, M.: Transitioning legacy assets to a product line architecture. ACM SIGSOFT Software Engineering Notes 24(6), 446–463 (1999)
25. Ebert, C., Smouts, M.: Tricks and Traps of Initiating a Product Line Concept in Existing Products. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), pp. 520–525 (2003)
26. Kircher, M., Schwanninger, C., Groher, I.: Transitioning to a Software Product Family Approach - Challenges and Best Practices. In: 10th International Software Product Line Conference, 2006, pp. 163–171 (2006)
27. Briggs, R.O., Kolfschoten, G.L., Vreede, G.J.d., Dean, D.L.: Defining Key Concepts for Collaboration Engineering. In: Americas Conference on Information Systems, AIS, Acapulco (2006)
28. De Vreede, G.J., Kolfschoten, G.L., Briggs, R.O.: ThinkLets: a collaboration engineering pattern language. International Journal of Computer Applications in Technology 25(2), 140–154 (2006)
29. Kolfschoten, G.L., Appelman, J.H., Briggs, R.O., de Vreede, G.J.: Recurring patterns of facilitation interventions in GSS sessions. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences, pp. 19–28 (2004)
30. Boehm, B.W., Ross, R.: Theory-W software project management principles and examples. IEEE Transactions on Software Engineering 1989 15(7), 902–916 (1989)
31. Harder, R.J., Keeter, J.M., Woodcock, B.W., Ferguson, J.W., Wills, F.W.: Insights in Implementing Collaboration Engineering. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS 2005, p. 15b (2005)
32. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Transactions on Software Engineering 29(3), 210–224 (2003)
33. Egyed, A.: A Scenario-Driven Approach to Traceability. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, pp. 123–132 (2001)
34. Barnard, J.: A new reusability metric for object-oriented software. Software Quality Journal 7, 35–50 (1998)
35. McCabe, T.J., Butler, C.W.: Design complexity measurement and testing. Communications of the ACM 32(12), 1415–1425 (1989)