Empir Software Eng
DOI 10.1007/s10664-007-9037-5

Empirical studies in reverse engineering: state of the art and future trends

Paolo Tonella · Marco Torchiano · Bart Du Bois · Tarja Systä

Editor: Daniel M. Berry

Abstract Starting with the aim of modernizing legacy systems, often written in old programming languages, reverse engineering has extended its applicability to virtually every kind of software system. Moreover, the methods originally designed to recover a diagrammatic, high-level view of the target system have been extended to address several other problems faced by programmers when they need to understand and modify existing software. The authors' position is that the next stage of development for this discipline will necessarily be based on empirical evaluation of methods. In fact, this evaluation is required to gain knowledge about the actual effects of applying a given approach, as well as to convince the end users of the positive cost–benefit trade offs. The contribution of this paper to the state of the art is a roadmap for the future research in the field, which includes: clarifying the scope of investigation, defining a reference taxonomy, and adopting a common framework for the execution of the experiments.

Keywords Reverse engineering · Taxonomy · State of art · Empirical framework

This work has been partially sponsored by Eureka Σ 2023 Programme; under grants of the ITEA project if04032 entitled Software Evolution, Refactoring, Improvement of Operational & Usable Systems (SERIOUS).

P. Tonella (✉)
ITC-irst, Centro per la Ricerca Scientifica e Tecnologica, Povo, Trento, Italy
e-mail: tonella@itc.it

M. Torchiano
Politecnico di Torino, Torino, Italy
e-mail: marco.torchiano@polito.it

B. Du Bois
University of Antwerp, Antwerp, Belgium
e-mail: bart.dubois@ua.ac.be

T. Systä
Tampere University of Technology, Tampere, Finland
e-mail: tarja.systa@tut.fi

1 Introduction

A software system is subject to changes throughout its lifetime. New requirements, e.g. requested new or changed functionalities and new constraints imposed to the software system, usually drive such changes. In fact, the largest proportion of software lifecycle costs is known to be due to maintenance activities (Sneed 1996; Jones 1998; Erlikh 2000). A requirement may be a new feature that is called for, a need to migrate a legacy system with its current functionalities to a new environment, or a need to fix the software. Thus, the reengineering and maintenance activities are usually task driven and can sometimes be hard to predict in advance.

Software maintenance and evolution are human-intensive activities, for which the process of cognition and knowledge acquisition play a central role. Several cognitive models of program comprehension have been defined in order to capture the mental processes and the comprehension strategies involved in program understanding, including the top–down, bottom–up, and integrated models (Brooks 1977; Soloway and Ehrlich 1984; Letovsky 1986; Littman et al. 1987; Pennington 1987; von Mayrhauser and Vans 1994).

Various reverse engineering and program comprehension tools and methods have been developed to help the software maintainer or reengineer understand the current structure and behavior of the subject system. Due to the task and environment-driven software maintenance and reengineering needs, it is, perhaps, not surprising that in many cases these tools have either been originally developed for, or their development has been greatly influenced by, a specific application example or software engineering team. This necessarily results in a set of greatly varying reverse engineering tools available, both in terms of their functionalities and visualization methods used. Guidelines for tool evaluation and selection are provided, for example, by ISO (1995). However, it is very difficult to compare the usefulness and usability of the tools in an objective and controlled way, especially when these are research prototypes, developed as proofs of concept, in support of novel methods or approaches.

Due to the goal-driven nature of reverse engineering and program comprehension activities, the analysis steps differ across a wide range of scenarios. Tools usually provide a set of functions for the software engineer to compose different views of the software, depending on the comprehension task at hand. While the variability of reverse engineering activities makes tool comparisons difficult, it is a challenge for also the tool developers. How to build a reverse engineering tool that would be generally usable, when not only the programming languages, but also the operations to be applied to analyze the software differ? In some tools this problem has been solved by providing a customizable and extensible tool that can be tailored to different needs, e.g., Rigi.[1] In software development, customization is often enabled using meta-models that are instantiated for certain purposes or domains. The possibility to customize and extend a tool clearly affects its usability and adaptability.

Reverse engineering tools are not widely adopted in industry (Zayour and Lethbridge 2001). Often, a tool is built to solve specific problems, and sometimes its scalability is not guaranteed. Integration and interoperability of reverse engineering

[1] http://www.rigi.csc.uvic.ca/

and forward engineering tools is also seldom achieved, because of the different underlying models and notations. Integration would, however, be desirable since reverse engineering activities are typically followed by forward engineering efforts. When tools are evaluated empirically, the experimental design should maximize the possibility of generalizing the observed efficiency and users' satisfaction to real industrial cases. Estimation of the effects of tool adoption beyond the learning curve is another central concern. In this respect, execution or replication of the studies in an industrial context would be a way to strengthen the findings.

When conducting empirical studies in reverse engineering, the variability of the skill and experience of the involved individuals may be high. Thus, subjects must be distributed carefully across the groups defined in the experimental design. Other sources of variability lie in the software development tasks given to the subjects, as well as the software systems themselves. These factors need to be taken into account and possibly controlled during the execution of the study.

State of the art studies in reverse engineering often do not comply to the prescriptions summarized above. Replication is not supported, industrial cases are rare and statistical significance is seldom proved. In order to help the discipline mature, we think that more systematic empirical evaluation is needed. The results of the individual studies should be gathered, in order to form a larger body of knowledge that can be used to support tool adoption. Moreover, the scientific relevance of an experience of tool usage can be assessed only if the validity and generalizability of the study are clearly described. The future of reverse engineering will be based on evidence about the proved effectiveness or ineffectiveness of certain methods and tools in given contexts. The aim of this paper is to provide a roadmap in order to guide the discipline toward this new stage of development of the research. Such a process was initiated by the Workshop on Empirical Studies in Reverse Engineering (WESRE) that was held in Budapest with STEP 2005.

The main contribution of this paper is a thorough discussion of the state of the art in reverse engineering research, followed by our proposal for future directions and guidelines for the discipline. We further suggest a list of criteria to characterize the currently available reverse engineering tools, and we propose the adoption of a general framework for tool evaluation and for the design of empirical studies.

2 Object of Study

The object of the empirical studies in which we are interested is *reverse engineering*. The most widely used and accepted definition of reverse engineering is that given by Chikofsky and Cross II (1990):

> *Reverse engineering is the process of analyzing a subject system to*
>
> - *Identify the system's components and their interrelationships and*
> - *Create representations of the system in another form or at a higher level of abstraction.*

To us, after more than a decade of research in reverse engineering, having in mind the huge variety of approaches, methods, techniques, and tools investigated so far, the definition above sounds quite limiting. The first part of the definition, identifying components and relationships, is narrowed to the problem of recovering

some design-level views of the system, representing its structure as a set of boxes and links, which are derived from the code disregarding some implementation details. Several available reverse engineering methods do no fit this definition. Consider, for example, slicing or feature location. None of the two produces an output consisting of components and relationships.

The second part of the definition by Chikofsky and Cross has a wider applicability and allows covering many more reverse engineering methods. However, it is not completely satisfactory, in that either it is too vague—what are the mentioned "representations of the system in another form?"—or it falls into the first case, if we interpret the "higher level of abstraction" as the design, assuming we are analyzing the code. Moreover, it misses a few important characteristics of reverse engineering. Namely, the context (i.e., the task in which reverse engineering is conducted), the role of automation (automated or semi-automated methods are in the scope of reverse engineering) and the knowledge acquisition process, which is an integral part of reverse engineering.

These problems with the definition above are partially addressed by Chikofsky and Cross, when they try to characterize reverse engineering in terms of its objectives (Chikofsky and Cross II 1990). Quoting from their paper, there are six key objectives in reverse engineering:

1. *Cope with complexity.*
2. *Generate alternate views.*
3. *Recover lost information.*
4. *Detect side effects.*
5. *Synthesize higher abstractions.*
6. *Facilitate reuse.*

Important elements of reverse engineering, such as its task-oriented nature and its information recovery purpose, are now mentioned. Of course, one cannot expect to be able to provide a formal or very precise definition: a common understanding of the discipline may result from different points of view, depending on the task, stakeholders, approach, etc., and it might be hard to formalize it in terms of an agreed definition. On the other hand, it is important to define the scope of the discipline, since it has reached a maturity level that demands empirical evaluation, and no empirical evaluation can be carried out without defining the precise scope of the studies.

Based on the paper by Chikofsky and Cross, as well as the reverse engineering works published afterwards, we give a characterization of the discipline, which does not attempt to be a precise definition, but at the same time delimits the scope of investigation of the empirical studies. To us, reverse engineering includes:

> *Every method aimed at recovering knowledge about an existing software system in support to the execution of a software engineering task.*

Knowledge acquisition means that in general multiple views and perspectives on a given software system are possible and potentially complement each other. Thus, there is no *hidden* or *lost* structure to be recovered—that there is hidden or lost structure to be recovered is a common misunderstanding of reverse engineering. Rather, a structure is superimposed in order to facilitate the execution of some software engineering task, e.g., maintenance or evolution. Structure imposition is

true also of design recovery, since in general multiple design views can be produced for a given system. The resulting structure may or may not be helpful depending on its end-users. Thus, the task-oriented nature of reverse engineering is necessarily complemented by a user-oriented characteristic. Moreover, usability issues are also central according to the definition above. In fact, the same piece of information recovered from the code may be immensely useful or completely unusable depending on the end user who is performing the current software engineering task and depending on the amount of knowledge the user already has about the system.

2.1 Reverse Engineering Methods, Techniques, and Tools

This section enumerates some of the reverse engineering methods, techniques and tools available in the literature, showing how they fit into the characterization of reverse engineering given above. Moreover, negative examples are also considered, in order to show what falls outside the scope of the empirical studies in reverse engineering considered in this paper.

To facilitate the description of methods, techniques, and tools, we distinguish between methods and techniques on one hand and tools on the other hand. A *method* or *technique* is a solution to one or more known problems in reverse engineering, e.g., design recovery. It might be argued that a method is usually more related to a process which involves humans in the loop while a technique is usually more focused on the automated part of such a process. As for distinguishing between a method and a technique, the standard glossaries for software engineering, e.g. IEEE 729 (IEEE 1983), do not allow a sharp distinction between "method" and "technique". Therefore, in the rest of this paper, we do not distinguish between a method and a technique and use only the word "method."

A *tool* is an application which implements and supports the usage of one or more methods. Clearly, each method can be implemented or supported in different ways by several tools.

Figure 1 shows the organization of a necessarily non-exhaustive sampling of contributions available in reverse engineering, according to the separation between methods and tools described above. Each method is associated with a specific reverse engineering problem. For example, *Slicing* aims determine how the value of a variable at a given statement is computed by the program under analysis. *Clone detection* aims to detect code fragments that were replicated and slightly changed. *Concept or feature location* aims to isolate the code portions responsible for the implementation of a given concept or feature. *Design recovery* aims to produce a structural view of a system's components and relationships.

Each method can be further specialized according to the specific approach chosen to address the target reverse engineering problem. For example, slices can be computed statically or dynamically, can be subjected to amorphous transformations, or can be determined for inputs satisfying given conditions. Clones can be obtained by comparing the Abstract Syntax Trees (ASTs), the metrics or the textual representations of the program entities.

Each tool supports or implements a method, making specific choices of interaction modes, user involvement, customizability, etc. Each such choice affects the effectiveness of the tool. Thus, the ability to address the initial reverse engineering problem

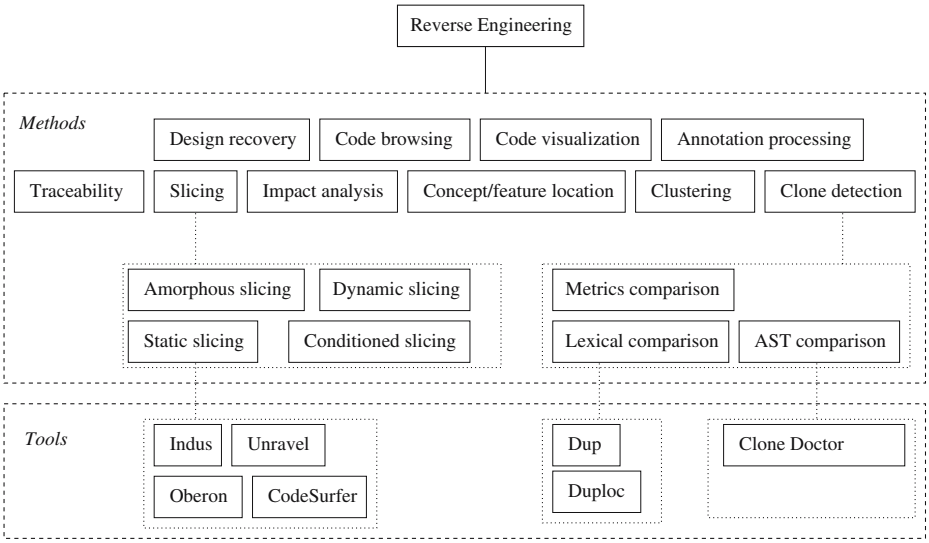

Fig. 1 Sample of reverse engineering methods and tools

depends on the selected method as well as the implementation embedded in a given tool.

By contrasting the methods and tools listed in Fig. 1 against the characterization of reverse engineering given in the main part of Section 2, we can notice that each satisfies the broader definition, although some of them, e.g., feature location, do not satisfy the original definition by Chikofsky and Cross. Every method or tool in Fig. 1 aims to support knowledge acquisition and is expected to be adopted for execution of software engineering tasks. In some cases, e.g., clone detection, the task dependence is quite apparent. In other cases, the method is quite general-purpose, but its effectiveness is dependent on the task at hand and on the already available knowledge. Finally, each of them deals with existing software systems. An example of a process falling outside the broader definition given above is testing, the aim of which is fault detection; knowledge aquisition might be only a side effect.

2.2 Taxonomy

When designing and conducting an empirical study in reverse engineering, its positioning in the field should be easily and clearly determined. In fact, the whole picture can be constructed incrementally only by execution of families of related experiments (Basili et al. 1999). This approach demands for the definition of a taxonomy of the methods and tools under investigation. Given such a taxonomy, it becomes possible to interpret the narrow results of a single study in the wider context. It is possible, for example, to determine the most closely related experiments or to design further experiments which cover neighboring areas. It is also possible to understand the generality of the results in terms of its height in the taxonomy.

Clearly, the definition of a common, agreed taxonomy upon of reverse engineering approaches requires a major effort of the research community, involving evaluation

and discussion of alternative proposals. This paper does not claim to provide any definitive solution to this problem, which is not the focus of the paper. However, the availability of a reverse engineering taxonomy is of fundamental importance for the next developments of the discipline, since it allows positioning and understanding the role of the empirical works. Thus, we intend to initiate a discussion on an agreed taxonomy for reverse engineering by presenting a provisional list of criteria to be adopted for the classification of the existing contributions.

The following list of criteria can be applied to classify the object of study in the execution of an empirical study in reverse engineering:

- *Method or Tool:* Is it a method or a tool?
- *Dynamic or Static:* Does it involve execution of the program or is it based on the source code alone?
- *Input:* What input, e.g., source code in a given programming language, executable program and execution scenarios, test cases, documentation, etc., does it require?
- *Output:* What output, e.g., diagrams, text, hypertext, facts populating a database, annotated code, etc., does it produce?
- *Interaction:* What interaction modes, e.g., navigation, queries, focusing, successive refinement, etc., does it support?
- *User Guidance:* What guidance, e.g., none (i.e., it is completely automated), manual evaluation of output, selection of appropriate inputs, definition of patterns, filtering, etc., does it require from the user?
- *Task Applicability:* To what tasks can it be applied? Is it general or special purpose?
- *Scalability:* To what system size can it be applied?

The authors of the present paper are actively contributing to start a collaborative effort for the definition of a reference reverse engineering taxonomy. The official presentation of the Wiki-style Web site hosting the taxonomy took place during the 2nd Workshop on Empirical Studies in Reverse Engineering (WESRE), held in Benevento, co-located with WCRE 2006 (Sim and Di Penta 2006). The taxonomy is open to contributions from all researchers working in the field, according to the Wiki-style of information gathering. The Web site for the taxonomy is: http://lore.cmi.ua.ac.be/reWiki.

3 State of the Art

In the software engineering field, which can be considered as the parent or a close relative of reverse engineering, there have been several attempts to systematize and provide guidelines for empirical studies (Wohlin et al. 2000; Kitchenham et al. 2002). Recognizing the similarity with social sciences, an initial suggestion (Singer 1999) was to use the American Psychological Association (APA) guidelines for describing experimental studies in software engineering. This approach has been further extended in the EU project ESERNET (Conradi and Wang 2003).

A number of papers have been focusing on the usage of evaluation methods (Tichy et al. 1995; Basili 1996; Zelkowitz and Wallace 1997; Shaw 2002). An overview of controlled experiments in software engineering has been proposed by (Sjoberg et al. 2005). A systematic characterization of software engineering

experiments has been proposed with the goal of building structured knowledge on the basis of experiment families (Basili et al. 1999). Aside from general surveys (Nelson 1996; Muller et al. 2000; Waters and Chikofsky 1994), we are unaware of any papers evaluating the usage of empirical studies in the more narrow domain of reverse engineering.

As a means to describe the current state of the art, we reviewed conference proceedings and journal volumes over the last 4 years, between 2002 and 2005. Our selection of four conferences and four journals is based on their relation w.r.t. the subject of reverse engineering.

The four selected conferences are:

- The Working Conference on Reverse Engineering (WCRE)
- The Conference on Software Maintenance and Reengineering (CSMR)
- The International Conference on Software Maintenance (ICSM)
- The International Symposium on Empirical Software Engineering (ISESE)

The four selected journals are:

- The Journal on Software Maintenance and Evolution: Research and Practice (JSME)
- The IEEE Transactions on Software Engineering (TSE)
- The ACM Transactions on Software Engineering Methodology (TOSEM)
- Empirical Software Engineering: An International Journal (EMSE)

In total, the considered proceedings and journals published 260 papers concerned with reverse engineering. A characterization of the type and purpose of these studies is provided in Tables 1 and 2 respectively. Table 3 shows the object of the studies. This classification was obtained by interpreting the papers' title, abstract and conclusion. Each paper was also skimmed to verify the categorization. A detailed description of the type and purpose (used in Tables 1 and 2) of the empirical studies is deferred to Section 4, where a classification and design framework is presented.

About one fourth, i.e., 64, of the papers discussing a method or tool matching our description of reverse engineering did not provide any attempt to evaluate their approach. These papers contributed a conceptual proposition or a proof of concept of their approach, and refrained from any form of experimentation. To discuss the use of evaluation methods in the remainder of this section, we exclude these papers. After this initial screening, the size of our review sample is limited to 196.

Among the papers in our selection which include an empirical evaluation, a large majority consists of experience reports and case studies—Rows 2 and 3 of Table 1; 139 papers or 70.9% of those involving any empirical evaluation. In other words, just

Table 1 Type of study of the 260 selected papers

Type	No. of papers	Percentage (%)
No experimentation	64	24.6
Experience report	70	26.9
Case study	69	26.5
Quasi-experiment	39	15.0
Randomized experiment	11	4.2
Observational study	7	2.6

Table 2 Purpose of the 260 selected papers

Purpose	No. of papers	Percentage (%)
Conceptual proposition	58	22.3
Proof of concept	94	36.1
Quantification	49	18.8
Comparison	49	18.8
Conditioned comparison	2	0.7
Review	2	0.7
Post-facto	6	2.3

one third of the papers evaluated their approach by controlling the setting, including quasi- and randomized experiments, as well as observational studies.

3.1 Objects of study

While classifying papers included in our review, we distinguish between (1) the tool used in the study, and (2) the method supported by the tool. In this subsection, we discuss the methods targeted in the works considered in our review and term these the objects of study.

Table 3 lists the objects of study and the number of papers covering each. Every reviewed paper was assigned a unique object of study, determined by its main focus. Each object falls into the characterization of reverse engineering provided in Section 2. To narrow the list, we do not show any object which was the object of study in less than five papers or articles. The lack of a widely accepted taxonomy makes it difficult to relate the different studies and to understand how they contribute to a method in particular and the field as a whole. As a result, the interpretation of the

Table 3 Objects of study

Object of study	No. of papers	Percentage (%)
Visualization	16	10.4
Design recovery	16	10.4
Slicing	14	9.1
Behavior recovery	13	8.5
Repository mining	13	8.5
Architecture recovery	9	5.8
Clustering	9	5.8
Impact analysis	8	5.2
Dependency analysis	8	5.2
Source recovery	8	5.2
Feature localization	6	3.9
Design pattern recovery	6	3.9
Traceability analysis	6	3.9
Clone detection	6	3.9
Concept localization	5	3.2
Design flaw detection	5	3.2
Information retrieval	5	3.2

object of study of each paper, performed during our review, might to some extent be questionable.

3.2 Population

In the studies we reviewed and which employed some type of empirical validation, 93.4% focused on the performance of automated tools, excluding the role of humans.

Clearly, there is the need for more studies in which the role played by the humans in the reverse engineering process is assessed together with the role of the tools. In fact, in the execution of any real maintenance task, tools can only assist humans, by augmenting and equalizing their skills, but tools can never replace humans.

3.3 Context

Around one fourth, i.e., 51, of the studies including an evaluation compared methods and therefore specified the conditions under which these were tested. We have categorized the factors, independent variables, included in these studies:

- *Comparing a tool or method against a placebo*, for example, use of program slicing for debugging.
- *Comparing different tools or methods*, for example, different clustering algorithms, feature identification methods, clone detection tools.
- *Comparing different configurations of a tool*, for example, the threshold used by an information retrieval method.
- *Comparing run-time performance in different contexts*.

Around half, i.e., 106, of the studies measured the performance of the method under study: quantification, comparison, and post-facto studies. The observed, dependent variables included in these studies are typically performance characteristics, e.g., precision and recall metrics, used to evaluate the identification effectiveness against a reference gold standard, and resource utilization metrics.

Again, the typically used variables reveal that the role of humans is underestimated. Most observations capture properties of the output produced by a tool, not its usefulness for the end user. The choices of dependent variables and of the kind of study are made with the aim of evaluating the intrinsic potential of the studied method, regardless of the way users take actually use the method.

4 Adopting an Experimental Framework

Several empirical studies in reverse engineering hold a large degree of similarity with soft or social sciences. The empirical branches of both hard and soft sciences have since a long time defined widely accepted standards for presenting and characterizing the result of their studies. Such guidelines make it easier to understand a research paper, since the reader knows where to look for the sought information. The search for the relevant studies is simplified, since papers can be classified more easily. The general validity or goodness of a paper can be assessed with a quick browse. It is easy to identify blind spots of the studies and to design families of experiments to cover them.

Based on the lack of a systematic organization and characterization of the studies available in the literature (see previous section), we propose the adoption of a framework, defined along the lines of similar frameworks investigated in the broader area of software engineering by other authors (Basili et al. 1999; Wohlin et al. 2000; Sjoberg et al. 2005). The framework can be used both to design and to classify empirical studies in reverse engineering. It consists of six main dimensions:

1. Type of the study,
2. Object of the study,
3. Purpose,
4. Focus,
5. Population,
6. Context.

In the following six subsections, each dimension is described in more detail.

4.1 Type of the Study

In the literature, there are many taxonomies of empirical studies. Partly based on (Zelkowitz and Wallace 1998), we identified three main factors that help characterize the type of study:

- *Multiplicity*: whether several cases are observed.
- *Control*: whether control has been exercised on the main factor and on the context factors and cofactors.
- *Randomness*: whether, in the presence of multiple cases, the assignment to the experimental groups was random or by convenience.

Such factors allow us to cover all the relevant studies found in the reverse engineering literature.

If only one or a few cases are observed, we have, depending on the level of control applied, one of two types of studies:

- *Experience report*: the treatment is applied to one case, but no specific effort is devoted to controlling the context. The goal is not insight, but rather to show superiority of the proposed technology (Zelkowitz and Wallace 1997). Accordingly, the set-up, data collection, and data analysis are not discussed in detail.
- *Case study*: the treatment is applied to one case, and the collection of data, targeting insight in the attributes of a set of products or processes, is discussed in detail, as are the set-up and data analysis. No variables are manipulated at different levels.

Such studies are the basis to achieve a higher maturity level.

When multiple cases are observed, we can distinguish whether or not control is applied and then, whether or not randomness is sought (Shadish et al. 2001):

- *Experiment*: the treatment is applied under control in order to observe the effects. To identify and isolate the effects of the treatment, also factors other than the treatment that may influence the outcome, should be controlled.

As a function of whether or not randomness is sought, it is possible to identify two sub-types:

(a) *Randomized experiment*: an experiment in which subjects are assigned to receive the treatment or an alternative condition by a random process.
(b) *Quasi-Experiment*: an experiment in which subjects are not assigned to conditions randomly, e.g. by convenience sampling or voluntary assignment.

- *Observational study*: a study that unobtrusively gathers observations in search for statistical support connecting factor and effect variables. Such a study often takes the form of a *survey*, in which random sampling is applied to select a population of cases to be observed (Pfleeger and Kitchenham 2001–2002).

In addition, we consider also *indirect* studies that are based on empirical results achieved in previous research:

- *Systematic review*: A systematic review is based on the evaluation and interpretation of all accessible pre-existing research relevant to a topic of interest (Kitchenham 2004). In such a study, the sampling of cases is performed by convenience, selecting those cases that have been studied in the past.

We include the *systematic review* type of studies instead of the more generic *review* since the former has a coded procedure while the latter is often used to describe a subjective, possibly commented, collection of literature. We distinguish between *systematic review* and *observational study* since the former is not directed toward supporting a specific factor–effect relationship, while this support is required for the latter.

4.2 Object of the Study

Objects are the entities under study in an empirical investigation (Wohlin et al. 2000). In the Reverse Engineering field, an object is usually a method or a tool (see Section 2). Unfortunately, in most cases, it is hard to separate the method from the tool, e.g., because the method has been implemented in a prototype tool that plays an essential role in the application of the method.

When a comparative study is conducted, there may be more than one object of the study, e.g., annotations and drawing editors. It is important to mention here how the object of the study is related to the treatment of the study. Typically, when there are multiple objects, the treatment is the type of object; in other cases, it may be the intensity or some other parameter of the object. In order to position a specific study in the available literature, a reference taxonomy for the object of a study, defined along the lines given in Section 2.2, should be adopted.

4.3 Purpose

It is possible to identify seven main purposes in the Reverse Engineering field:

- *Conceptual Proposition* (CP): The authors propose a concept, process, or theory. The paper provides a theoretical contribution, but there is no demonstration of feasibility nor of evaluation.

- *Proof of Concept* (PC): The authors demonstrate the feasibility of the proposal. There is no evaluation of any measurable property of the proposal. At most, qualitative insight in the strengths and weaknesses of the proposal are derived.
- *Quantification* (Q): The authors quantify a property of the approach with regard to a certain task, e.g., analyzing the recall and precision of an information retrieval method.
- *Comparison* (C): The authors clarify quantitative differences between properties of the proposal and at least one alternative.
- *Conditioned Comparison* (CC): The authors clarify quantitative differences between properties of the proposal and at least one alternative in at least two conditions, e.g., method A is better than method B in condition $C1$, but the opposite is true in condition $C2$.
- *Review* (R): The authors present a literature overview.
- *Post-facto* (PF): The authors review existing data and make a determination about a correlation between two measurements.

4.4 Focus

There are two fundamentally different foci of the empirical studies in tool support for software engineering (Walenstein 2002):

- *Usefulness*: the actual assistance provided by the object of the study to the developer in fulfilling her task.
- *Usability*: the actual ease in learning and applying the object of the study.

The usability of software engineering tools has not been extensively researched, and the same holds for reverse engineering tools (Lethbridge and Singer 1997). Usability testing methods, initially developed for consumer products or non-professional usage, should be applied critically (John and Marks 1997). Often, the comparisons have been carried out in an ad hoc manner, without a proper scientific test, e.g., with no control on randomness or on co-factors. The general metrics of usability as defined by ISO 9241-11 (ISO 1998)—effectiveness, efficiency and subjective satisfaction—can be adapted to the Reverse Engineering field. *Effectiveness of a tool* measures the fraction of each problem that can be solved by using the tool, and *efficiency of a tool* measures amount of effort needed to solve any given reverse engineering task. *Subjective satisfaction using a tool* measures each professional user's satisfaction while using the tool in his or her daily work over a long period of time. In addition, *organizational satisfaction with a tool* is an organizational generalization of subjective satisfaction using the tool. The realization of these attributes in a software tool differs significantly from the usability metrics of consumer products, for which the emphasis is often on learnability, intuitiveness, aesthetics, and short-term satisfaction and delight by an individual user.

One central decision of usability studies is whether the study is conducted in a controlled laboratory environment, or whether a more ethnographic or contextual approach is chosen. In reverse engineering tool research, combinations of both are needed.

Nielsen (1993) defines usefulness to consist of usability and utility. He defines utility as follows: "utility is the question of whether the functionality of the system in principle can do what is needed, and usability is the question of how well users

can use that functionality." Usability interacts with utility when reverse engineering tools are evaluated. Poor usability may be confused with low utility, whenever the information recovered by the method implemented in the tool is by itself useful, but it is made unusable in practice by poor design of the tool's interface with the user. A usability study allows researchers to discriminate between the two, getting a better picture of the reasons for the observed performance.

4.5 Population

In the studies in which a multiplicity of cases is observed, there is an important bifurcation of the type of studies. In general we can have two types of subjects of the study that define the dimension to which statistical methods are applied:

- *Humans*: the object of the study is applied by *several persons*, e.g., programmers or maintainers, to perform a given task on *one or more programs*.
- *Programs*: the object of the study is applied manually or automatically to perform a given task on *several programs*.

Equally important is to characterize the population: e.g. for humans, students vs. professional programmers; for programs, the relevant features include size, application domain, programming language and paradigm, platform, etc. Such a characterization is fundamental when it comes to generalizing the results of the study, that is, to obtain external validity.

4.6 Context

In order to establish the external validity of the study and the applicability of the tested method or tool, it is essential to extensively document the context of the study. In the literature, only a few works consider the main factors influencing maintenance and comprehension activities. This section gives a short overview of them, with the aim of providing an initial checklist, which could be used by researchers to identify the most important co-factors in their studies.

- *Ontology of software maintenance*
  Kitchenham and others propose an ontology (Kitchenham et al. 1999) of factors that might affect the result of empirical studies in software maintenance productivity, quality or efficiency. The ontology addresses factors grouped into four areas:

  (a) *Maintained product*: (1) size, (2) application domain, (3) age, (4) maturity, (5) composition, and (6) quality.
  (b) *Maintenance activities*: (1) quality, type, and age of input and output artifacts, (2) human, hardware and software resources, (3) product history, (4) modification size and criticality, (5) kind, (6) requirements involved, (7) configuration and event management, (8) change control, and (9) development technology, paradigm, and procedure.
  (c) *Maintenance organization processes*: (1) maintenance events and performance targets of service level agreements, (2) formality and quality of configuration management, (3) factors influencing the maintained product, (4) investigation reports, and (5) modification activities.

(d) *Staff*: (1) attitude, (2) motivation, (3) skill, (4) roles of the organization staff, (5) number and types of users, (6) goals, and (7) employers of customers.

- *Ontology of program comprehension strategies*
  Storey and others propose an ontology (Storey et al. 1997) of cognitive issues which should be considered during the design of a software exploration tool. They are grouped into three categories:

  (a) *Maintainer characteristics*: (1) application domain knowledge, (2) programming domain knowledge, (3) maintainer expertise, (4) creativity, and (5) familiarity with the program and the programming environment.
  (b) *Program characteristics*: (1) application domain, (2) programming domain, (3) program size, (4) complexity and quality, (5) availability of documentation, and (6) programming environment.
  (c) *Task characteristics*: (1) task type, size and complexity, (2) time constraints, and (3) environmental factors.

Precise description of the context in which a study was performed is fundamental in order to understand the scope of applicability of the reported results. A researcher may borrow categories and terms from these two ontologies when describing the context of his or her study.

4.7 Empirical Study Datasheet

We encourage the *good practice* of providing a datasheet for empirical studies. The datasheet's structure, derived from the proposed framework, is shown in Table 4.

In addition, the authors of an empirical study in reverse engineering should provide pointers to the raw data sets and a complete replication package. The raw data allow the readers of the study to replicate the statistical analyses and to conduct further analyses. The replication package shall contain all the details required to replicate the experiment, including but not limited to:

- Sampling procedures;
- A complete description of each task;
- A full description of the software environment.

The statistical and practical relevance of the study should also be mentioned explicitly. At least the statistical significance, e.g., p-value, and the effect size should be included. These values are essential to enable the readers combining the results from several studies, e.g. by meta-analysis (Miller 2000), and obtaining stronger evidence related to a method or tool (Pickard et al. 1998).

Table 4 Empirical study datasheet

Item	Content
Type	Multiplicity, control, randomness
Object	Technique and/or tool
Purpose	{*CP, PC, Q, C, CC, R, PF*}
Focus	Usefulness, usability
Population	{humans, programs} + description
Context	Confounding factors

Adoption of a common datasheet, made available on the Web and possibly included in publications, would support a better understanding of the results, their comparison against alternatives and replication of the experiment. Overall, it would be a step forward in order to bring the discipline to maturity.

5 Research Directions

The Technology Acceptance Model (TAM), originally developed by Davis (1989), models users' acceptance of technology at work as a combination of perceived ease of use, perceived usefulness and intention to use, which leads to usage behavior. Venkatesh and Davis (2000) have extended this model by describing factors affecting perceived usefulness. Such factors include: (1) subjective norms, (2) image, (3) job relevance, (4) output quality, (5) result demonstrability, (6) the user's level or experience, (7) whether or not the user volunteered, (8) perceived ease of use, and (9) usage behavior.

In order for an organization to be able to efficiently exploit new tools, the factors of technology acceptance must be fulfilled. Usability is one of the central contributors, but in an organization, the implications of software developer team dynamics—as suggested by the TAM model—need to be studied. It is beyond the scope of this paper to propose the exact procedure for such studies, but an ethnographic approach would be required to reveal the degrees of organizational acceptance and individual acceptance.

After several decades of research, the discipline of reverse engineering aims at having an impact on the practice of software development, maintenance, and evolution. To this aim, empirical evidence is expected to play a major role. This requires the definition of a research agenda for the next developments in the field. In this paper, we examine the main areas that need further investigation and tried to summarize the state of the art. The directions for future investigation can thus be organized along the aspects of the empirical studies in reverse engineering that are lacking in one or more respects:

- *Taxonomy refinement*: One aim for conducting empirical studies in reverse engineering is building knowledge from families of related experiments (Basili et al. 1999). This knowledge construction process requires the capability to position each individual study within a taxonomy for describing the objects of a study. This paper proposes a set of criteria for the definition of such a taxonomy. We expect that these will be discussed and refined in future research.
- *Agreed upon framework*: Design and classification of empirical studies in reverse engineering should adhere to an agreed upon framework, in order to support comprehension, comparison, and replication. Based on the existing literature, this paper summarizes the common elements proposed in the past into a tentative framework that can be used with reverse engineering studies. We expect to see this framework refined and adjusted in the future. Most importantly, we will contribute to its diffusion and wide adoption, in order to make the studies conducted in this field more uniform.
- *Benchmark*: Empirical studies in reverse engineering have a strong dependence on the programs being analyzed and on the maintenance tasks being executed. To support controlled replication, a benchmark of programs, associated with a set of maintenance tasks to be executed on them, is essential. With this

benchmark, variability can be reduced, and comparisons and replications become easier to conduct. Software engineering benchmarks have been recognized as one of the challenges for the research in the field (Sim et al. 2003). They represent the operationalization of a paradigm emerging in a discipline and their success depends on several sociological and technical factors (Sim et al. 2003). Also their relationship with experiments and case studies was investigated (Sim et al. 2003). Actually, a few examples of benchmarks do already exist in reverse engineering, such as *Xfig* (Sim and Storey 2000; Sim et al. 2000). We envisage an increasingly widespread usage of similar benchmarks.

- *Replication*: Every empirical study in reverse engineering should be accompanied by a replication kit (also known as laboratory package (Shull et al. 2002, 2004)), to allow any researcher to replicate the study in a different environment following the instructions contained in it. Public availability of replication kits is essential to move the discipline toward maturity. Moreover, a standard form for replication kits would be extremely useful. Data repositories aimed at supporting replication of software engineering studies are already available in the areas of metrics, predictive models (Sayyad-Shirabad and Menzies 2005), and testing (Do et al. 2005). They could serve as a good starting point for the definition of a reverse engineering data repository.
- *Datasheets*: The organization of the descriptive data produced for each empirical study should follow a common template. Section 4 proposes a skeleton for such a template, in the hope that it will be adopted and refined. Reading the results, understanding the experimental design, assessing the validity, and extracting useful lessons from published studies would become much easier if datasheets were organized according to a standard format. This datasheet is part of the infrastructure that is needed by the discipline, and it is sketched in Section 4 of this paper.

6 Concluding Remarks

In this paper, we try to delimit the scope of investigation of the empirical studies in reverse engineering, by discussing thoroughly the object of study. Moreover, we review the recent publications in the field, so as to have a clear picture of the state of the art. Based on the review, we recognize the need for a reference taxonomy and propose a tentative set of taxonomic criteria to support the positioning of each study in the wider context, and we describe a tentative framework, borrowed from the broader area of software engineering, which should guide researchers conducting new studies.

The vision that solicited this study and that emerged at the Workshop on Empirical Studies in Reverse Engineering (Budapest, with STEP 2005) is the future scenario of the research in this area. The next stage of the research method will involve obtaining empirical evidence that benefits outweigh the costs. The obtention of this evidence is expected for two reasons. On one hand, claims made in the field that are based on only personal trials or anecdotes cannot be accepted any longer. Rigorous empirical evaluation is required to produce relevant contributions that advance the field. On the other hand, the field has reached a maturity level in which it can have a concrete impact on the practice of software development and maintenance. However, this impact can be achieved only if the effects of the field's adoption are carefully studied, in oder to identify the most cost-effective approaches.

In this paper, we try to sketch the future trends in reverse engineering by providing a research agenda, which we consider an essential step to move forward.

Acknowledgement The authors are grateful to Associate Editor Daniel Berry, who greatly contributed to the improvement of this manuscript.

References

Basili VR (1996) The role of experimentation in software engineering: past, current, and future. In: ICSE '96: Proceedings of the 18th international conference on software engineering. IEEE Computer Society, Washington, DC, USA, 1996. ISBN 0-8186-7246-3, pp 442–449

Basili VR, Shull F, Lanubile F (1999) Building knowledge through families of experiments. In: IEEE transactions on software engineering. IEEE Computer Society, pp 456–473

Brooks R (1977) Towards a theory of the cognitive processes in computer programming. Int J Man-Mach Stud 9(6):737–741

Chikofsky E, Cross II J (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw 7(1):13–17

Conradi R, Wang AI (eds) (2003) Empirical methods and studies in software engineering, experiences from ESERNET. Springer, Berlin Heidelberg New York

Davis FD (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quart 13(3):319–340

Do H, Elbaum SG, Rothermel G (2005) Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empir Softw Eng 10(4):405–435

Erlikh L (2000) Leveraging legacy system dollars for e-business. In: IEEE IT Pro, pp 17–23

IEEE (1983) Glossary of software engineering terminology. Technical Report

ISO (1995) ISO/IEC DIS 14102: evaluation and selection of computer-aided software engineering (CASE) tools

ISO (1998) ISO 9241: ergonomic requirements for office work with visual display terminals (Part 11. Guidance on Usability)

John B, Marks S (1997) Tracking the effectiveness of usability evaluation methods. Behav Inf Technol 16(4–5):188–202

Jones TC (1998) Estimating software costs. McGraw Hill

Kitchenham B (2004) Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University and NICTA

Kitchenham B, Travassos G, von Mayrhauser A, Niessink F, Schneidewind NF, Singer J, Takada S, Vehvilainen R, Yang H (1999) Towards an ontology of software maintenance. J Softw Maint 11(6):365–389

Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734

Lethbridge T, Singer J (1997) Understanding software maintenance tools: some empirical research. In: Proceedings of the IEEE workshop on empirical studies in software maintenance (WESS), pp 157–162

Letovsky S (1986) Cognitive processes in program comprehension. In: Soloway E, Iyengar S (eds) Proceedings of the 1st workshop on empirical studies of programmers. Ablex Publishing Company, Norwood, NJ

Littman DC, Pinto J, Letovsky S, Soloway E (1987) Mental models and software maintenance. J Syst Softw 7(4):341–355

Miller J (2000) Applying meta-analytical procedures to software engineering experiments. J Syst Softw 54(1):29–39

Muller H, Jahnke J, Smith D, Storey M-A, Tilley S, Wong K (2000) Reverse engineering: a roadmap. In: Proc. of the 22nd international conference on software engineering, future of software engineering track. ACM Press, New York, pp 47–60

Nelson ML (1996) Software maintenance cost estimation and modernization support. Technical Report, Old Dominion University

Nielsen J (1993) Usability engineering. Academic, Boston, MA

Pennington N (1987) Comprehension strategies in programming. In: Olson G, Sheppard S, Soloway E (eds) Proceedings of the 2nd workshop on empirical studies of programmers. Ablex Publishing Company, Norwood, NJ

Pfleeger SL, Kitchenham B (2001–2002) Principles of survey research – parts 1–5. ACM Software Engineering Notes 26,27(6,1,2,3,5):16–18,18–20,20–24,20–23,17–20
Pickard LM, Kitchenham BA, Jones PW (1998) Combining empirical results in software engineerig. Inf Softw Technol 40(1):811–821
Sayyad Shirabad J, Menzies T (2005) The PROMISE Repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada
Shadish WR, Cook TD, Campbell DT (2001) Experimental and quasi-experimental designs for generalized causal inference. Houghton Mifflin, Boston, MA
Shaw M (2002) What makes good research in software engineering? Int J Softw Tools Technol Transf 4(1):1–7. ISSN 1433-2779 (print), 1433-2787 (electronic)
Shull F, Basili VR, Carver J, Maldonado JC, Travassos GH, Mendonça MG, Fabbri SCPF (2002) Replicating software engineering experiments: addressing the tacit knowledge problem. In: Proc. of the international symposium on empirical software engineering (ISESE). IEEE Computer Society, Los Alamitos, CA, pp 7–16
Shull F, Mendonça MG, Basili VR, Carver J, Maldonado JC, Fabbri SCPF, Travassos GH, de Oliveira MCF (2004) Knowledge-sharing issues in experimental software engineering. Empir Softw Eng 9(1–2):111–137
Sim SE, Di Penta M (eds) (2006) Proceedings of the 13th working conference on reverse engineering. IEEE Computer Society, Benevento, Italy, 2006. ISBN 0-7695-2719-1
Sim SE, Easterbrook SM, Holt RC (2003) Using benchmarking to advance research: a challenge to software engineering. In: Proceedings of the 25th international conference on software engineering. IEEE Computer Society, Los Alamitos, CA, pp 74–83
Sim SE, Storey M-AD (2000) A structured demonstration of program comprehension tools. In: Proceedings of the working conference on reverse engineering (WCRE), pp 184–193
Sim SE, Storey M-AD, Winter A (2000) A structured demonstration of five program comprehension tools: lessons learnt. In: Proceedings of the working conference on reverse engineering (WCRE), pp 210–212
Singer J (1999) Using the American Psychological Association (APA) style guidelines to report experimental results. In: Proceedings of the IEEE workshop on empirical studies in software maintenance (WESS), pp 71–75
Sjoberg D, Hannay J, Hansen O, Kampenes V, Karahasanovic A, Liborg N, Rekdal A (2005) A survey of controlled experiments in software engineering. IEEE Trans Softw Eng 31(9):733–753
Sneed H (1996) Encapsulating legacy software for use in client/server systems. In: Proceedings of the working conference on reverse engineering, Monterey, CA, pp 104–119
Soloway E, Ehrlich K (1984) Empirical studies of programming knowledge. IEEE Trans Softw Eng 10(5):595–609
Storey M-AD, Fracchia FD, Mueller HA (1997) Cognitive design elements to support the construction of a mental model during software visualization. In: Proceedings of the 5th international workshop on program comprehension (IWPC), p 17
Tichy WF, Lukowicz P, Prechelt L, Heinz EA (1995) Experimental evaluation in computer science: a quantitative study. J Syst Softw 28(1):9–18
Venkatesh V, Davis FD (2000) A theoretical extension of the technology acceptance model: four longitudinal field studies. Manage Sci 46(2):186–204
von Mayrhauser A, Vans A (1994) Comprehension processes during large scale maintenance. In: Proceedings of the international conference on software engineering, Sorrento, Italy. IEEE Computer Society Press, Los Alamitos, CA, pp 39–48
Walenstein A (2002) Cognitive support in software engineering tools: a distributed cognition framework. Ph.D. thesis, School of Computing Science, Simon Fraser University
Waters RG, Chikofsky E (1994) Reverse engineering: progress along many dimensions. Commun ACM 37(5):22–25
Wohlin C, Runeson P, Host M, Ohlsson MC, Regnell B, Wesslen A (2000) Experimentation in software engineering: an introduction. Kluwer, Boston, MA
Zayour I, Lethbridge T (2001) Adoption of reverse engineering tools: a cognitive perspective and methodology. In: Proc. of the international workshop on program comprehension (IWPC). IEEE Computer Society, Los Alamitos, CA, pp 245–258
Zelkowitz M, Wallace D (1997) Experimental validation in software engineering. Inf Softw Technol 39(11):735–743
Zelkowitz M, Wallace D (1998) Experimental models for validating technology. IEEE Computer 31(5):23–31

Paolo Tonella is a senior researcher at ITC-irst, Trento, Italy where he leads the Software Engineering research unit. He received his laurea degree cum laude in Electronic Engineering from the University of Padua, Italy, in 1992, and his PhD degree in Software Engineering from the same University, in 1999, with the thesis "Code Analysis in Support to Software Maintenance". Since 1994 he has been a full time researcher of the Software Engineering group at ITC-irst. He participated in several industrial and European Community projects on software analysis and testing. He is the author of "Reverse Engineering of Object Oriented Code", Springer, 2005. His current research interests include reverse engineering, AOP, empirical studies, Web applications and testing.

Marco Torchiano is an assistant professor in the Department of Control and Computer Engineering at Politecnico di Torino, Italy; previously he has been post-doctoral research fellow at Norwegian University of Science and Technology (NTNU), Norway. He received an MSc and a PhD in Computer Engineering from the Politecnico di Torino. He participated in several EU and national research projects and served in the program committee of several international conferences. He recently organized the 2nd International Workshop on Empirical Studies in Reverse Engineering. His current research interests are: empirirical software engineering, OTS-based development, and software engineering for mobile and wireless applications. He published more than fourty research papers in international journals and conferences. He is co-author of the book "Software Development - Case studies in Java" from Addison-Wesley and is co-editor of the book "Developing Services for the Wireless Internet" from Springer.

Bart Du Bois is a postdoctoral researcher in the Lab On Reengineering at the Department of Mathematics-Computer Science, University of Antwerp, Belgium. He received his Master's Degree (2002) and Ph.D (2006) in Computer Science from the University of Antwerp, Belgium. His research interests include reverse engineering and reengineering of large evolving software systems, with a particular focus on software quality.

Tarja Systä is a professor at Tampere University of Technology, Institute of Software Systems. She received her MSc degree in mathematics from the University of Turku in 1992 and a PhD degree in computer science from the University of Tampere in 2000. Her current research interests include software maintenance and analysis, software architectures, model-driven software development, and development and management of service-oriented systems. She has published several international journal and conference articles on these topics, and she actively reviews for software engineering journals and conferences.