Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 47 (2005) 1019–1032
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Simulating families of studies to build confidence in defect hypotheses

Forrest Shull [a,*], Daniela Cruzes [b,c], Victor Basili [b], Manoel Mendonça [c]

[a] Fraunhofer Center—Maryland, 4321 Hartwick Road, Suite 500, College Park, MD 20740, USA
[b] Department of Computer Science, University of Maryland, College Park, MD 20742, USA
[c] Computer Networks Research Group (NUPERC), Salvador University (UNIFACS), Rua Ponciano de Oliveira, 126 Salvador, BA 41950-275, Brazil

Available online 17 November 2005

Abstract

While it is clear that there are many sources of variation from one development context to another, it is not clear a priori what specific variables will influence the effectiveness of a process in a given context. For this reason, we argue that knowledge about software process must be built from families of studies, in which related studies are run within similar contexts as well as very different ones. Previous papers have discussed how to design related studies so as to document as precisely as possible the values of likely context variables and be able to compare with those observed in new studies. While such a planned approach is important, we argue that an opportunistic approach is also practical. The approach would combine results from multiple individual studies after the fact, enabling recommendations to be made about process effectiveness in context.

In this paper, we describe two processes with which we have been working to build empirical knowledge about software development processes: one is a manual and informal approach, which relies on identifying common beliefs or 'folklore' to identify useful hypotheses and a manual analysis of the information in papers to investigate whether there is support for those hypotheses; the other is a formal approach based around encoding the information in papers into a structured hypothesis base that can then be searched to organize hypotheses and their associated support. We test these processes by applying them to build knowledge in the area of defect folklore (i.e. commonly accepted heuristics about software defects and their behavior). We show that the formal methodology can produce useful and feasible results, especially when it is compared to the results output from the more manual, expert-based approach. The formalized approach, by relying on a reusable hypothesis base, is repeatable and also capable of producing a more thorough basis of support for hypotheses, including results from papers or articles that may have been overlooked or not considered by the experts.

1. Introduction

Empirical studies have long been used to provide confidence in assertions about what is true and not true in the software engineering domain. By providing rigorous observation of the effects of a development technique under specific conditions, empirical study allows an analysis of the conditions under which practices yield similar effects on a project's cost, quality, or schedule. Where different results are obtained under different contexts, the contexts can be analyzed to hypothesize about which variations led to the differences in results[1].

The ability to build up rigorous abstractions of information about practices not only provides confidence in individual assertions about specific techniques, but also is an important capability in providing an engineering basis for software development. This capability is an essential part of the experience factory approach [4], for example, as well as more recently suggested in the idea of evidence-based software engineering [14].

Due to the large number of possible variations from one development environment to another, we have argued [5] that this process of knowledge building about practices must therefore, based on families of related studies, designed so that a range of context variations can be explored. Although this approach is logically appropriate, it does pose some practical problems. First, it is not always clear a priori what the important context variables are, meaning that important sources of variation may go unmeasured. Second, because there are so many potential context variables, we cannot design experiments that cover all of them and may not be able to identify environments, which offer coverage of all the variables. In other words, to build an effective family of

* Corresponding author.
E-mail addresses: fshull@fc-md.umd.edu (F. Shull), daniela@unifacs.br (D. Cruzes), basili@cs.umd.edu (V. Basili), mgmn@unifacs.br (M. Mendonça).

[1] The set of potential context variables is quite large, but includes for example such issues as: team size (larger teams require more communication overhead, making certain practices more or less effective); team experience (certain practices may require a level of skill development that makes them better suited for experts); lifecycle model (if a project is contractually required to follow a waterfall lifecycle then practices that better fit a spiral lifecycle may not be appropriate); etc.

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.09.007

studies, multiple experimenters, without having a clear concept of all the contributing factors, must agree a priori on a set of variables to collect and identify environments that cover the complete set of variables, so that all studies are fully comparable.

Providing robust decision support for software development—i.e. making a statement about what development practices can help achieve goals related to cost, quality, or schedule for a given environment—requires the collection of data across multiple contexts so that we can begin to elicit these variables.

To make this approach work in practice, and build a suitably large and varied dataset, we need to leverage existing work wherever possible. That is, we have to be able to mine information about the relevant variables and the effect of practices from previous studies that were not a priori designed to fit together. In effect, this requires the ability to simulate a family structure over independent studies that were not explicitly designed to build directly on one another.

In this paper, we extend the work in our earlier paper [5] by presenting a process for creating a 'post hoc' family of studies. We present a set of results in the area of software defect phenomenology to demonstrate the feasibility and practicality of this approach.

2. Challenges in combining studies post hoc

Designing a replicated study that fits together with previous work to build a family of studies is difficult (cf. [7,19]). A mistake in experimental design or application of the practice under study can easily render the study invalid and waste potentially hundreds of person—hours of effort from multiple subjects.

Combining the studies after the fact also suffers from several difficulties (although at least if mistakes are made, only the final body of knowledge is affected and only the analysis needs to be redone).

Borrowing concepts from work on ontology integration [15,31], we can see that mismatches between empirical studies are the key type of problems that hinder the combined use of independently developed studies.

Visser et al. enumerate the types of mismatches that may occur at the semantic-level, which describe differences in the way a domain is modeled. Semantic-level mismatches can arise when two or more papers that describe (partly) overlapping domains are combined. Based on classifications by Visser and Klein, the different types of mismatches include:

- Conceptualization mismatches: a conceptualization mismatch is a difference in the way a domain is *interpreted* (conceptualized), which results in different concepts or different relations between those concepts. There are two types:
  - Scope: in this type of mismatch, two results seem to represent the same concept, but do not have exactly the same meaning (although there may be some overlap). For example: two studies may refer to the 'cost' of a practice, although one may include only the cost of the effort to apply the practice in the calculation, while the other may include the start-up costs as well (e.g. sending personnel for training).
  - Model coverage and granularity: this type of mismatch describes problems that can arise in trying to combine results when it is unclear to what part of the domain those results are applicable. For example, a study may make claims about a large class of software development projects while only having evidence concerning one or two specific instances of such projects.
- Explication mismatches: an explication mismatch is a difference in the way the conceptualization is *specified*. This can manifest itself in mismatches in definitions, mismatches in terms and combinations of both. There are three types:
  - Synonymous terms: synonyms, in this context, are different terms which refer to the same concept. A trivial example is the use of the term 'strength' in one study and the term 'cohesion' in another, to refer to the same concept (that is, the amount of interaction within components of a system) . Although the technical solution for this type of problems seems relatively simple (the use of thesauri), the integration of studies with synonyms or different languages usually requires significant human effort to resolve the semantic issues.
  - Homonym terms: this type of mismatch occurs when the meaning of a term is different in different contexts. For example, the term 'interface defects' can have different interpretations, depending on the context: it can refer to a defect in the human–computer interface or a defect in the interfaces between two software components. This inconsistency is much harder to handle; human knowledge is required to solve this ambiguity. One must also be careful that a mismatch in the explication is not masking a deeper mismatch at the conceptual (scope) level.
  - Encoding: values in the studies may be encoded in different formats. For example, a numbers of lines of code may be represented as 'KLOC' or as 'LOC' or 'SLOC,' etc. The first step is to check that the basic definition of a line of code is the same across each source, i.e. that the underlying entity being counted is really the same in each case. If so, and the differences are in fact that the results have just been reported using different

levels of granularity, a transformation step or wrapper can simply be applied to eliminate all those differences.

To avoid mismatch problems within this paper, we will use the following IEEE definitions [12] for defects and related phenomena. Where necessary, we annotate direct quotes from the papers cited for data to enforce a consistent terminology that was not always used by the individual authors.

- Error: a defect in the human thought process made while trying to understand given information, to solve problems, or to use methods and tools;
- Fault: a concrete manifestation of errors within the software (note that one error may cause several faults and various errors may cause identical faults);
- Failure: a departure of the operational software system behavior from users' expected requirements (a particular failure may be caused by several faults and some faults may never cause a failure).

Where necessary we will also use the term defect as a generic term, to refer to an error, fault, or failure.

3. Building hypotheses using empirical evidence.

The current state of the practice for abstracting information from across several studies is to perform a literature search, reviewing the relevant literature in a rigorous way and constructing a textual summary of the evidence related to a given issue. If the sources do not agree then it is the reviewer's responsibility to construct a fair summary of the evidence on both sides of the issue. We have been using the concept of 'folklore' as a way to focus such a literature search in a way that is useful for combining data and producing well-formed and empirically-supported hypotheses [1]. We define folklore as informal, subjective lessons learned based on the experiences of people in the field.

This approach involves considering the folklore as a kind of hypothesis and identifying papers that either support or countermand the folklore. In studying a set of related papers, the researcher is able to build evidence for or against the folklore, to refine the folklore into testable hypotheses, or to recognize the context variables that differentiated when and where the hypotheses were true or false (thus, creating more specific sub-hypotheses).

In this section, we present an example of folklore testing as an example of how testable hypotheses can be generated by such a literature search. We choose the area of software defects as a topic, which is especially rich in folklore; this is not surprising, as debugging software defects is an activity that consumes a significant amount of time for most software developers. Two pieces of folklore that we identified are:

1. There are patterns in the defect classes found in classes of projects.
2. The vast majority of defects are interface defects.

Both items of folklore are important as together they state the basis for much of what is done in defect analysis. If there are patterns in the defects we find in projects, then analyzing the defects in a project in a particular environment will allow us to better understand what techniques, methods and processes to apply in that environment in the future. If the majority of defects are interface defects, then the kinds of quality assurance techniques applied need to have the goal of preventing or identifying interface defects. To keep the example short we make use of only three papers, with which we happen to have significant experience:

- Endres75: one of the earliest papers on software defects, it describes the release of a version of an operating system in which approximately 500 modules (140 K source lines of assembly code) were affected by the modification. (The 140 K total program size had to be calculated based on average values given for the individual modules, providing an example of resolving an encoding mismatch as defined in Section 2). Defects were classified as being problem-specific, implementation-specific, or textual-specific. In this analysis, problem-specific defects are considered as due to a misunderstanding of the requirements [9].
- Weiss/Basili85: this paper deals with a study of three projects in the software engineering laboratory (SEL) at NASA/GSFC. They all dealt with development of requirements, design and code for ground support software for unmanned spacecraft control, and were 50–120 K source lines of Fortran code. One of the issues specifically explored in the study was the distribution of defects by error origin (i.e. according to the phase in which the misunderstanding took place). In this case the phases recorded were requirements, functional specification, design or coding of multiple components, design or coding of a single component, language, environment, and other [32].
- Basili/Pericone84: this paper deals with the development and evolution over three years of a general purpose program for satellite planning studies in the SEL. The system was 90 K source lines of Fortran code and the requirements kept changing and evolving over time. The same question was asked in this study as Weiss/Basili85 but the phases were defined slightly differently. Phases here were requirements, functional specification, design, coding, and evolution over 3 years [3].

The Endres75 study only had one project, but the largest source of defects (46%) in that project was associated with a misunderstanding of the problem domain. In the Weiss/Basili85 study, all three projects had very similar profiles but the majority of defects (between 56 and 72%) occurred during the design or coding of a single component. In the Basili/Pericone84 study, even though data was gathered in the same SEL environment, the majority of defects were caused by misunderstanding of the requirements (55%).

The difference in results among these studies allows us to examine the impact of the context variables. In Weiss/Basili85, the organization had earlier developed many similar projects,

so that although the requirements varied, the organization had experience in developing that type of system. Thus, the fact that the majority of the defects were occurring in the coding phases was due to the fact that the requirements and high-level design were relatively well understood by the developers but new hires were used to develop the code. On the other hand, the Endres75 and Basili/Pericone84 studies involved projects with relatively new requirements and less understanding of those requirements by the organization. This allows us to postulate two new hypotheses:

1a. In novel projects, the largest source of defects will be due to misunderstanding of the requirements [Supported by: Endres75, Basili/Pericone84 data sets; Contradicted by: None].
1b. In projects where the organization has built up experience in the application, the largest source of defects will be coding [Supported by: Weiss/Basili85 data set; Contradicted by: None].

This analysis also allows us to recognize that the hypothesis as originally stated is also supported.

With regard to the second hypothesis, the issue is the distribution of interface defects. Endres75 reported that only 15% of the defects found were interface defects. However, he defined interface defects based on the number of components changed, i.e. a defect is an interface defect if more than one component (module) must be changed to fix the problem. We can think of this as an 'implementation interface defect'. Using the same definition, Basili/Pericone84 found that only 11% of the defects were 'implementation interface defects'. However, Basili/Pericone84 used a second definition of interface defect, associated with the number of components examined; i.e. it is an interface defect if more than one component (module) had to be examined in order to design the change. We can call this, a 'design interface defect' (The multiple definitions for 'interface defects' are an example of the homonyms, as well as a scope mismatch, as discussed in Section 2). This is a much more inclusive definition and most definitions would lie somewhere between these two definitions. Using this latter definition, Basili/Pericone84 found that 39% of the defects were 'design interface defects.' Thus, in either case the folklore is not true, meaning that a useful hypothesis describing defect behavior should be formulated regarding the exactly opposite condition:

2′ Interface defects will not comprise the vast majority of defects in a system [Supported by: Endres75, Basili/Pericone84; Contradicted by: none].

These examples demonstrate the benefit of accumulating results from multiple studies in multiple papers, recognizing how context variables can provide insights into hypothesis refinement as well as open new hypotheses. However, this approach also demonstrates the difficulties alluded to in Section 2:

- The lack of formality makes it easy to miss mismatches among studies. In the analysis above, the integration of studies was done by someone with a large degree of context knowledge concerning all of the studies described, but this cannot always be expected to be the case.
- The analysis is not reusable; literature reviews are typically done to address a particular issue and if a related but different issue is of interest to another researcher, then typically the analysis must be redone from scratch.
- Due to the textual nature of the review, it is hard to create a summary that is both rigorously backed up by evidence and that summarizes the current state of knowledge.

4. A more formal methodology

Based upon the results from the folklore-based method of abstracting empirical results, we recognized the need for a more effective and rigorous process for identifying possible hypotheses based not just on folklore but on various forms of information from the unrelated studies themselves. In this section, we describe the process we created, which combines a structured approach to knowledge gathering plus data mining techniques. The outputs of this method are compared to those in Section 3 as a way to check the efficacy of this new method.

As in the folklore-based approach, this methodology has the goal of building a set of hypotheses that represent the knowledge about software development practices contained in multiple studies, which need not have been designed specifically to produce related data. This knowledge is represented as hypotheses to reflect its provisional nature, that is, each study in a family can only contribute evidence for or against some statement about software development phenomena. No amount of studies ever proves such a relationship outright.

As input, the methodology requires a focus of study, i.e. a (set of) software engineering phenomenon(a) about which information is needed.

The process consists of five steps (Fig. 1), starting with a definition of the problem area, then the selection of papers of interest, the extraction of information from these papers, integration of the information and finally the analysis and interpretation of the results. This process is iterative, in that the results of a given step may convince the researcher to go back to a previous step and redo the associated activities. For example, if the researcher is not satisfied with the information extracted from a set of papers, he or she may use these results to

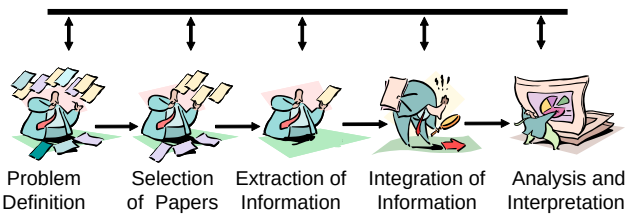

Fig. 1. Building and effective set of structured information—the meta process.

suggest new areas to search in order to select more papers for analysis.

The output of the process is a set of conclusions and new knowledge that arises from the process. As a secondary output, the process creates a structured hypothesis base, a structured and searchable repository of issues about which information has been found. The advantage of the creation of the structured hypothesis base is that it can be reused. Other researchers can evolve and reuse it according to new research goals as they arise.

The first two steps of the methodology, defining the problem and selecting relevant papers, are performed much the same as they would be in any method, no matter how formal, and are thus not discussed here at length. Defining the problem of course depends largely on the interests of the researcher. The problem definition is also related to the amount of knowledge already accumulated in an area. For example, as more evidence is accumulated we can move from studying how failure-prone software products are to which types of failures are most common; to which types of products display common failure profiles; to which context variables make those failure types more likely to occur. This allows us to evolve our knowledge into more useful models over time.

The second step, selecting papers that can be searched for evidence in the focus area, is also conducted largely in the same way regardless of the individual process being followed. To be suitable, a paper must provide some empirical information or experience-based hypotheses relating to the focus of study.

The remaining steps will be performed in a quite specific way in this methodology, and so are described in more detail in the following subsections.

4.1. Extracting information from papers

In this step, the researcher must review each paper, looking for potential hypotheses described in the text. According to Gay, a hypothesis is a tentative explanation for certain behaviors, phenomena, or events that have occurred or will occur. 'A good hypothesis states as clearly and concisely as possible the expected relationship (or difference) between two variables and defines those variables in operational, measurable terms' [10].

Any hypothesis should be stated in such a way that data can be collected that either supports or refutes the hypothesis. For the purpose of our analysis, we classify the hypotheses as tested and untested. A tested hypothesis is a tentative explanation for certain behaviors, phenomena, or events that have occurred in an empirical study. An untested hypothesis (otherwise called a belief or assumption) is a tentative explanation for certain behaviors, phenomena, or events without explicit reference to empirical data.

While reviewing the papers selected in the previous step, the researcher should highlight the important information (so that there can be some traceability to the original source if questions arise later). After highlighting the information, it is important that key details are transferred to data entry forms to create the structured base for analysis. For now, we are using forms implemented in Excel, although in future work we intend to create a tool to support the activities of the process. The researcher should focus his or her search specifically on the following categories and types of information: hypotheses, context descriptions, and definitions.

Hypotheses. It is important to note that in this step, all relevant information should be highlighted, without requiring critical appraisal from the researcher regarding the quality of support. For our structured hypothesis base, we want to collect beliefs and assumptions as well as hypotheses with empirical results that support or reject their contention. These various levels will be distinguished from one another by using the 'support' field described below. The reason for this practice is that we want to be able to describe the possible relationships among variables as broadly as possible, drawing from the common knowledge of experienced professionals as well as the subset of relationships that have been empirically verified. However, at the same time, we want to carefully record the quality of that support so that the end results can be understood in relation to the associated level of confidence that is warranted.

Our experience is that the section of the paper describing the analysis of the empirical data is where most tested hypotheses can be found. The conclusions are a good place to find hypotheses, although these are many times repetitions from earlier in the text. Some hypotheses will also be found in tables and figures; although not explicitly stated in the text of the paper, relationships that are expressed visually for readers will need to be translated into textual form to be inserted into the hypothesis base in a usable way.

The following information should be recorded on the appropriate form for each hypothesis:

- Plain text: the hypothesis should be stated in such a way that data can be collected that either supports or refutes the hypothesis. A good hypothesis states as clearly and concisely as possible the expected relationship (or difference) between dependent and independent variables and defines those variables in operational, measurable terms. The hypothesis should be written using exactly the wording from the paper so that traceability is assured.
- Original source of hypothesis: a reference to the paper in which the information was found.
- Level of support: the support should be described as one of the following levels:
    - Significantly positive: the results support the hypothesis and the results of a test are included to show that the results are statistically significant, that is, with a high degree of certainty are not the result of pure chance.
    - Positive: the results in the paper support the hypothesis, but no significant statistical results can back this up.
    - Null: the hypothesis has been tested but the results in the paper neither support nor contradict the hypothesis.

  - Negative: the results in the paper contradict the hypothesis, but no significant statistical results can back this up.
  - Significantly negative: the results contradict the hypothesis and the results of a test are included to show that the results are statistically significant, that is, with a high degree of certainty are not the result of pure chance.
  - Belief: the hypothesis is formulated based on assumption or belief but has not been tested.
- Confidence in support: reflects the level of confidence that can be placed in the results and author's analysis. It is a 4-level scale:
  - High: results were rigorously measured and in a representative industrial environment.
  - Med: measurement was not completely rigorous or the context was not realistic.
  - Low: measurement was not completely rigorous and the context was not realistic.
  - None: no evidence was presented (The hypothesis describes a belief and has not been tested).
- Observations: this is a free-text field where the researcher can keep track of additional information that is important for correctly understanding or interpreting the hypothesis.

Context descriptions: this form holds details concerning the environment from which the measures were drawn. There will be at least one record on this form associated with each source (possibly more, if the paper describes data that was collected from several projects). Our experience is that the context descriptions usually come shortly after the introduction. As different studies report different metrics of interest to them, not every paper will have all of the following information. However, the template should be filled out as completely as possible given the information that has been published. Missing values will need to be accounted for during the analysis, as they limit the strength of the conclusions that can be drawn.

The attributes of the context description form are:

- Source: a reference to the paper in which the information was found.
- Criteria related to the development team:
  - Number of Subjects: number of subjects from which the author collected the information to do the analysis.
  - Experience of subjects: level of skill or expertise, captured using whichever measures the authors described in the paper.
- Criteria related to the software product:
  - Project size: size of the project, described using whichever measures the authors described in the paper.
  - Degree of complexity of the application: complexity, described using whichever measures the authors described in the paper.
  - Notation: if applicable, the language used to encode the model of the system that was the object of study. For example, in a study of code the notation would be the programming language used; in a study of requirements, the notation could be natural language or a formal modeling notation like SCR.
  - Application description: a brief description of the application. For example: 'The development of an on-board flight control program for a new aircraft.'
  - Development phase: the development phase in which the information was gathered: requirements, architecture, design, code, test, maintenance.
- Criteria related to the data collection process:
  - Data source: the source of the data, like forms, compilers, tools, etc. The main intent is to provide information during the analysis phase concerning how objectively or subjectively the data was reported. Each source may have multiple sources, for different types of information; the granularity required in the form depends on the source.
  - Collection period: number of months for which the collection was done and the year that the information was collected.
- Observations: this is a free-text field where the researcher can keep track of additional information that is important for correctly understanding or interpreting the hypothesis.

The context description can be extended with other fields, specific to the given focus, that help the analysis of hypotheses. For example, using the focus of 'Defects' we gathered information including:

- Number of defects: number of analyzed defects;
- Number of modules: number of analyzed modules.

Definitions. The extraction of all the terms used in the paper is important for the interpretation and analysis of the structured hypothesis base. To address the challenges regarding mismatches among different sources, described in Section 2, the definitions from each paper must be recorded in a systematic way and then reconciled, to remove the chance that misinterpretations occur during the analysis of results.

There should be one record on this form for each measure collected in each source. Our experience is that the definitions are found in papers usually following the introduction.

The attributes of the definition form are:

- The term itself, for example: module.
- The description, as given in or implied from the text. For example: module is a named subfunction, subroutine, or the main program of the software system.
- The original source: a reference to the paper.

The hypotheses, context descriptions and definitions are important for comprehending the possible measures and influencing factors for the focus of study. We are also working on ways of recording less directly connected contextual information, like software development challenges and lessons learned about empirical studies, that are also important to

gather from the papers because they help the researcher to understand the context of the paper and of the conclusions. However, this work is ongoing and outside the scope of the current paper.

4.2. Integrating information from papers

Information integration consists of two steps: (1) Integration of definitions; and (2) Formalization of the hypotheses based on the unified definition set. These steps may be iterated as needed.

The integration of definitions includes the creation of one unified definition schema, by finding the places in the definitions where they overlap, relating concepts that are semantically close via equivalence and subsumption relations (aligning) and checking the consistency, coherency and non-redundancy of the result [15,31]. The alignment of concepts is especially difficult, because this requires understanding of the meaning of concepts, but this step has been explored in ontological research, and many techniques and tools exists to facilitate its automation [20].

The unified definition schema has the main goal of solving problems of conceptualization and explication mismatches pointed in Section 2. Some of them will remain unsolved, and the researcher has to be aware of them before drawing conclusions from the analysis.

In the hypothesis formalization step, the hypotheses recorded from papers are formalized in such a way that structured analysis can be done. For the purposes of such analysis, we define hypotheses as being relationships between exactly two variables (one dependent and one independent). In cases, where the hypotheses as stated in the source paper describes multiple relationships, the formalized hypotheses have to be divided into several smaller ones of two variables each.

The formalization of hypotheses is an important step for a better analysis of the results. It is important that the researcher, when executing this step, keeps in mind the focus of the study, because this will direct the choice of the dependent and independent variables. We have found, in our experience, two main types of hypotheses: A statement about a logical relationship between two variables (e.g. 'As module size increases, the number of defects injected increases') and a statement of a mathematical function that relates two variables (e.g. 'Interface defects are expected to account for approximately 25% of all defects in a system'). This abstraction of hypothesis types is based on the hypotheses uncovered in our work to date; we do not claim that there may not be others, less often used, which we simply have not encountered yet. However, these types seem to fit equally well with information coming from case studies (which look at detailed relationships over the life of a single project) as well as controlled experiments (which investigate whether a treatment, such as using a new process, has any impact on some success criteria).

A formalized version of either type has three major components: A dependent variable, an independent variable, and a relation between them. The variables themselves can be any entity discussed in the hypothesis (e.g. 'module size,' 'number of injected defects,' 'number of interface defects').

Each of the dependent and independent variables also has three additional fields which may be left blank: the direction of effect, the magnitude of effect, and constraints on the effect. These fields will only have values if the hypothesis contains appropriate information. In the case where the hypothesis describes a logical relationship, the direction and magnitude fields for each variable capture information about effects related to changes in value. For example, if the hypothesis is 'as module size increases by 10%, the number of defects injected increases by 25%' then the independent variable of 'module size' has the direction of 'increases' and the magnitude of '10%.' In cases where the hypothesis describes a mathematical function, the direction and magnitude capture the functional calculation. For example, if the hypothesis is '25% of all system defects will be interface defects,' then the dependent variable of 'interface defects' has the magnitude of '25%' but no direction.

A constraint on either variable simply captures information about the range of applicability of the variable. For the hypothesis 'as the size of C modules increases by 10%…', for example, the constraint field for the independent variable would contain the information that this is for the C programming language.

The relation field always describes the hypothesized relationship between the independent and dependent variables. Values can be: $\sim =$(approximately equal), $<$, $>$, $\leq$, $\geq$,$=$, $\neq$, $\Rightarrow$ (implies), $\neg \Rightarrow$ (does not imply).

As illustration, consider the following examples:

1. Plain text: larger modules are more complex than smaller modules
   a. Formalization:
      i. Increased size in modules implies increased complexity
         1. Independent direction: increased
         2. Independent magnitude of difference: N/A
         3. Independent variable: size
         4. Independent context: modules
         5. Effect: imply
         6. Dependent direction: increased
         7. Dependent magnitude of difference: N/A
         8. Dependent variable: complexity
         9. Dependent context: N/A
   b. Source: [3]
   c. Level of support: significantly positive
   d. Confidence: high
2. Plain text: 89% of defects can be corrected by changing only one module
   a. Formalization:
      i. 89% of defect corrections equal changes in one module
         1. Independent direction: N/A
         2. Independent magnitude of difference: 89%
         3. Independent variable: defect
         4. Independent context: module corrections
         5. Effect: equal

6. Dependent direction: N/A
7. Dependent magnitude of difference: N/A
8. Dependent variable: changes
9. Dependent context: within one module

b. Source: [3]
c. Level of support: significantly positive
d. Confidence: high

3. Plain text: there is a higher defect rate in smaller sized modules
   a. Formalization:
      i. Decreased size implies increased defect rate
         1. Independent direction: decreased
         2. Independent magnitude of difference: N/A
         3. Independent variable: size
         4. Independent context: N/A
         5. Effect: imply
         6. Dependent direction: increased
         7. Dependent magnitude of difference: N/A
         8. Dependent variable: defects
         9. Dependent context: rate
   b. Source: [3]
   c. Level of support: significantly positive
   d. Confidence: low
   e. OBS: the low confidence comes from the way that the authors explain the results: 'The most plausible explanation seems to be that the large number of interface [defects] spread equally across all modules is causing a larger number of [defects] per 1000 executable statements for smaller modules. Some tentative explanations for this behavior are that: the majority of modules examined were small, causing a biased result; larger modules were coded with more care than smaller modules because of their size; and [defects] in smaller modules were more apparent. There may still be numerous undetected [defects] present within the larger modules since all the 'paths' within the larger modules may not have been fully exercised' [3].
4. Plain text: errors in understanding interfaces and requirements are more difficult to correct than others
   a. Formalization:
      i. Requirements errors imply increased difficulty of defect correction
         1. Independent direction: N/A
         2. Independent magnitude of difference: N/A
         3. Independent variable: requirements
         4. Independent context: errors
         5. Effect: imply
         6. Dependent direction: increased
         7. Dependent magnitude of difference: difficulty
         8. Dependent variable: defects
         9. Dependent context: correction
      ii. Interface errors imply increased difficulty of defect correction
         1. Independent direction: N/A
         2. Independent magnitude of difference: N/A
         3. Independent variable: interface
         4. Independent context: errors
         5. Effect: imply
         6. Dependent direction: increased
         7. Dependent magnitude of difference: difficulty
         8. Dependent variable: defect
         9. Dependent context: correction
   b. Source: [32]
   c. Level of support: significantly positive
   d. Confidence: low

After the formalization of the hypotheses we have a structured hypothesis base that can now be used in a process of analysis.

4.3. Analysis and interpretation

We argue that different types of analysis (top–down or bottom–up) can be done depending on the researcher's interests and how well-specified the problem can be. If, during the problem definition phase, the researcher can only state a general topic of interest rather than a specific hypothesis, he/she can do a bottom–up analysis, using the data from the papers analyzed to specify the set of important context variables. On the other hand, if the researcher already has detailed hypotheses in mind, with influencing factors already identified, he/she can do a top–down analysis by searching the structured hypothesis base for any evidence describing those influence factors and their effects.

The input for the analysis is the structured hypothesis base, also containing the specified definitions and context descriptions. The output of the process depends on the researcher's goals for the analysis, which may include: new hypotheses with supporting evidence, new beliefs that reflect expert opinion and may not have been previously recognized as such, or the refutation or confirmation of a set of initial hypotheses or folklore.

We assume that in some way the researcher can follow a core of structured steps to do the analysis. We propose here only one possibility of these steps, but recognize that the exact process followed is dependent upon the specific researcher doing the analysis:

1. Filter papers, if desired, selecting a subset that is most interesting for the current analysis. (For example, the researcher may select a subset of papers in which he/she has the most confidence in the results.) As filtering may introduce bias, we recommend this be done only according to objective criteria, such as including only papers in peer-reviewed journals. Any filtering should be reported clearly in the report of the study results, to allow the rationale to be subjected to peer review.
2. Filtering based on level of support and confidence. The researcher can filter the base, for example to remove hypotheses with the support level of 'belief' (i.e. removing items with no empirical backing), giving more confidence in the results.
3. Organize the hypotheses that are still included in the analysis according to the dependent variables that they describe.

4. Choose a dependent variable to explore. This choice is based on the focus of analysis described in the definition of the problem.
5. Visualize influencing factors on the dependent variable. In this step the researcher can see which independent variables are related to the chosen dependent variable, and he/she can choose to explore some of these relations in more detail, for example, getting a sense if there is consensus about the direction of the relation, the magnitude of the effect, or the constraints under which those results are observed.
6. Based on context descriptions and definitions, certify that all mismatch problems are solved and that different hypotheses are not being combined. Filter problems not solved or hypotheses that cannot be combined to draw conclusions.
7. Draw conclusions based on the patterns observed, create new hypotheses, or refute or confirm initial hypothesis or folklore.

The overall process is iterative, and the researcher can choose to traverse the steps in an order influenced by her/his specific goals, e.g. by doing first, a bottom–up analysis to construct a detailed hypothesis, and then a top–down analysis to evaluate the body of evidence available in support of it.

While applying the process, the researcher may need to transform the values of information stored in the structured hypothesis base, as described in Section 2 (e.g. expressing measures in KLOC in LOC instead, or estimating the total size of a program from the sizes of its component modules). This is done in an ad hoc manner, depending on the hypotheses that the researcher is trying to abstract, and so may be largely dependent upon the particular context. Some of the translations may be stored in the structured hypothesis base if they appear likely to be reusable, but many will be too context-specific for this to be useful.

The bottom–up analysis considers only information found in papers published about the focus of study. That is, categories of phenomena and resulting hypotheses are built up based only on the information found in the literature search. The analysis of the structured hypothesis base is used to identify knowledge that cuts across multiple studies, for example in identifying variables that have an effect on the outcome.

The top–down approach considers that users may have intuition, folklore, or hypotheses; in short, some pre-existing opinions about which types of information and phenomena are interesting enough to collect information on. In this case, the analysis of the structured hypothesis base yields a refutation or confirmation of these preconceptions.

Many visualization techniques can be applied to facilitate the exploration of the structured hypothesis base [13,21]. In the next section we present a detailed example of applying the process in the area of software defects, in which we used Treemaps [29,30] to do the data exploration. Treemaps are a space-constrained visualization of hierarchical structures. They assign the available hypotheses into regions on the display, which allows researchers to see the hierarchies in datasets, in this context for example, to see the entire set of hypotheses organized first according to common dependent variables (i.e. specific aspects of the general topic of interest), and secondly, in additional subsets around common independent variables (i.e. the factors that influence those dependent variables).

5. A bottom–up analysis case study

5.1. The problem definition

To allow a comparison between our more formal methodology and the example given of the folklore-based analysis in Section 3, as a topic of study we select software defects.

We follow a two-pronged approach to compare the results of the methodology to those of other approaches. First, we use a relatively small set of papers in which one of the authors had some experience in the application context, to investigate whether or not the results that could be obtained from applying the methodology are useful compared to an expert's opinion of the contribution of the source articles. These results are described in Sections 5.2 and 5.3. Secondly, we reran the analysis on a larger set of papers done by other authors. In Section 5.4, we compare the results from our formal approach on this set against an external expert's manual analysis of the knowledge that could be built across studies, to assess the completeness and repeatability of our approach when done without relying on a deep knowledge of the application context of the source articles.

5.2. Results from a bottom-up analysis

To allow comparison with the folklore-based approach described in Section 3, we first apply the formal methodology using only the same three papers that were used in Section 3: Endres75, Basili/Pericone84, and Weiss/Basili85. The methodology was applied by one of the authors who did not have any direct experience with the papers. As was emphasized in our discussion of building up bodies of knowledge from families of studies, each study in the family need not be a 'strict' replication of one another, i.e. may have a different overall design and data collection strategy [5].

Sixty-seven plain text hypotheses were extracted from these papers. These were translated into 76 formalized hypotheses in the hypothesis base. During the analysis, we used only the hypotheses that were supported by empirical data, and we filtered the analysis to include only the relations that were supported by more than one paper. We used the Treemap tool[2] to do the data mining exploration on data.

We first created a hierarchy based on dependent variable, and found three large groups of hypotheses related to defects, changes, and effort. Looking in detail at the variable of 'defects' we created a new level of the hierarchy based on independent variable. Analyzing Fig. 2, we can see the main independent variables related to defects: interface, size, effort, misunderstandings of specification, changes, and others (In Fig. 2, each of the small rectangles represents an individual hypothesis, with

[2] http://www.cs.umd.edu/hcil/treemap/

the text shown inside the block. Different shading represents the different source papers from which the hypotheses are drawn. The larger groupings of hypotheses collect together all hypotheses about a common independent variable).

At this stage in the analysis, then, we have a systematic list of potential relations on the focus of interest (software defects), which may or may not represent new information to the researcher. Each relationship may be explored in more detail to examine the context in which the relationship held and the level of accumulated supporting data. Choosing to analyze 'interface' as independent variable yields the results shown in Fig. 3. We can see that two of the three papers fail to support the hypothesis that most defects are interface defects.

To try to solve this discrepancy, we looked in more detail at the definitions of 'interface' in each paper. In the two papers that do not support the hypothesis, the definition of interface defects is based on the number of components changed, i.e. a defect is an interface defect if more than one component (module) must be changed to fix the problem. In the third paper, interface defects are associated with structures existing outside the module's local environment. We can say then that, using the first definition, interface defects are not in fact the majority of the defects in this study.

Exploring the remaining independent variables, we could draw some additional conclusions:

A. Interface defects are not the majority of defects, using the definition of interface defects that is based on the number of components changed, i.e. a defect is an interface defect if more than one component (module) must be changed to fix the problem [Supported by: Supported by: Basili84, Weiss85 and Endres75; Contradicted by: none].
B. Misunderstandings of specifications make up the majority of defects when the developers are not experienced with the application domain [Supported by: Basili84 and Endres75; Contradicted by: Weiss85].
C. Size, alone, is not a factor to determine defect-proneness [Supported by: Basili84, Endres75, and Weiss85; Contradicted by: None].

Comparing these hypotheses to the results of the folklore-based approach, we see that Hypothesis A above matches exactly to the refined Hypothesis 2′ in Section 3. Hypothesis B above also maps directly to Hypothesis 1a in Section 3. Hypothesis C above is a concrete instance of the general case described in Hypothesis 1 (namely, that there are consistent patterns that can be found describing software defect occurrences).

5.3. An expanded hypothesis base

One of the advantages of the formal approach is that the structured hypothesis base is reusable and expandable as needed. To demonstrate, we can easily rerun the above analysis by adding additional papers into the base.

Specifically, we next added defect data from the following papers, which brought the total number of hypotheses (including belief statements) to 223:

1. Rombach/Basili87: this study was conducted in the maintenance environment of a major computer company in commercial systems [24].
2. Selby/Basili88: a single release of a code library tool [26].
3. Selby/Basili91: a single release of a code library tool [27].
4. Mashiko/Basili97: a set of four projects dealing with communication software [18].

Again, this large number of hypotheses was severely filtered for this example, by removing belief statements which did not

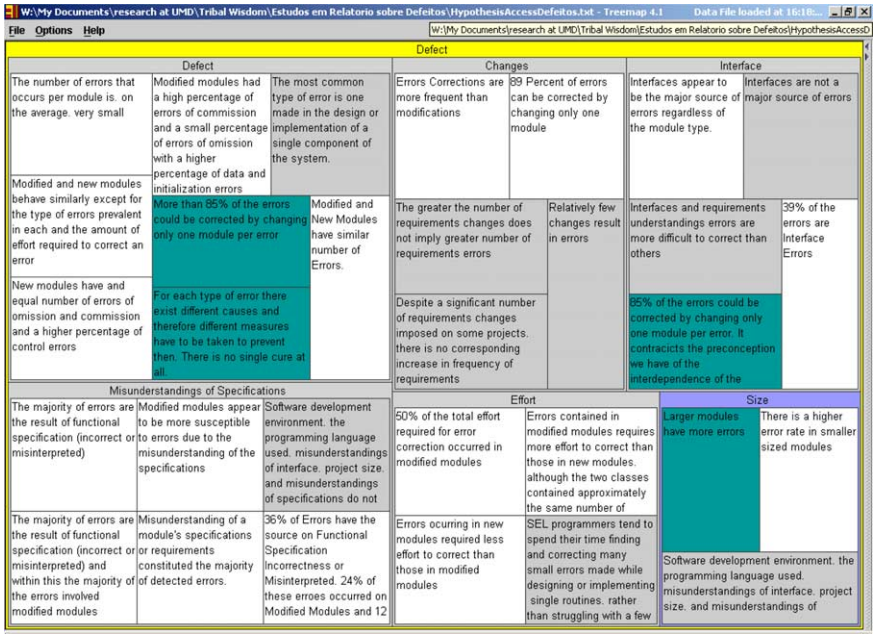

Fig. 2. Hypotheses about six independent variables affecting software defects.

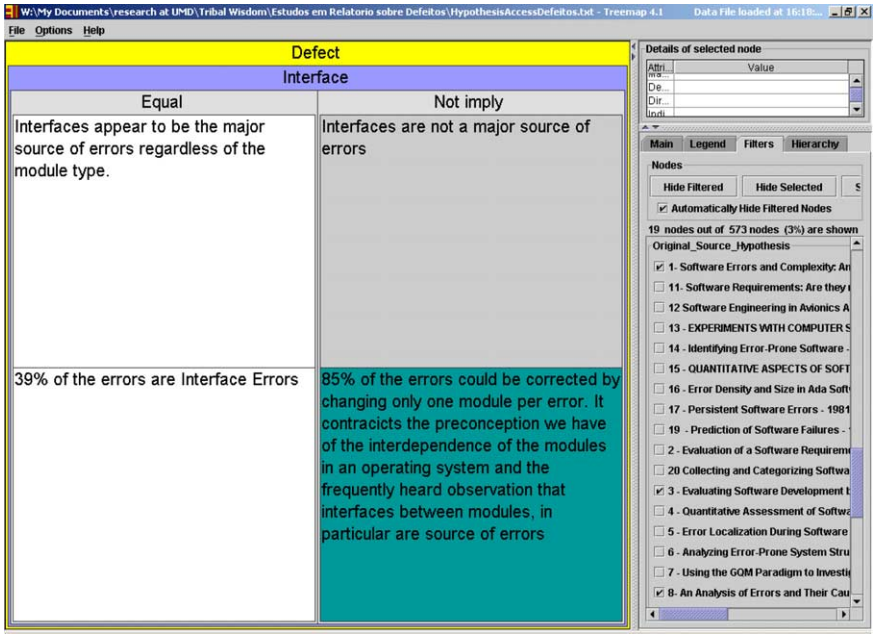

Fig. 3. Analyzing specific relationships between interfaces and software defects.

have empirical data addressing them and limiting analysis only to those hypotheses which were addressed by more than one paper.

The analysis process helps to get an understanding of the solution space covered by these studies. For example, while Mashiko/Basili97 has some hypotheses about interfaces and defects, upon resolving the semantic mismatches it becomes apparent that all of them are related to external interfaces (i.e. faults in the interface between the product and its external system) or human interfaces, thus covering quite separate phenomena from the other three papers which draw conclusions about internal interfaces (faults in the interface between modules).

The analysis of the remaining papers added no new hypotheses to the list about misunderstandings of specification, but increased the level of confidence in the hypothesis related to size by providing new hypotheses from additional contexts that matched the initial conclusions.

5.4. Using papers of other authors

To demonstrate that this methodology can use papers from other authors, we decided to replicate the study done by Brian Marick [17], in which several different defect databases were reviewed in order to formulate statements about common issues in defect behaviors. Specifically, we added to our hypothesis base defect data from the following papers [2,6,8,11,16,22,23,25,28,33] (It is important to note that these papers were selected to match the analysis done by Marick and not to produce a comprehensive and up-to-date body of defect knowledge).

- Basili81: the example application is an evaluation of the first stage of the A-7 flight software redevelopment: the production of the A-7 requirements document.
- Bell76 contains an analysis of two contexts. The first was a project in a graduate software engineering class, in which students wrote a set of software requirements and a preliminary system design for a student employment information system (SEIS). The second case was a large (1 million machine instructions) real-time Ballistic Missile Defense (BMD) system being developed with a top–down approach.
- Dniestrowski78: the data reported in this paper is about an avionics digital flight control for a FALCON 20 variable stability aircraft. The source code consisted of: 800 variables declarations, 1715 constant declarations, 3615 assembly instructions, 3230 assembly instructions, 1050 assembly directives, 9100 documentation.
- Glass81: operational software systems for military aircraft use was analyzed in this paper. Project A involved 150 programmers at the peak person-load, and contains about a half million instructions in the operational software alone. Project B involved 30 programmers and about 100,000 instructions.
- Lipow79: data from three large software projects are examined and an analysis of the effect on defects of certain preventive and detective tools and techniques is presented.
- Ostrand84: in this paper, they reported the results obtained from collecting defect data for nine months from a special-purpose editor project. The editor's source is about 10,000 lines of high-level language and assembly. The program

design and coding were done by three programmers over ten months, after the initial specification had been completed. The implementation represents approximately 18 persons-months of effort.

- Potier82: the software used in the experiments belongs to a family of compilers for the real-time language LTR II developed for a wide range of target machines. These compilers written in the LTR language consist of a common kernel specific to the LTR language and independent of the target machine, and a code generation part dependent on the target machine. The total number of lines is 64,939, with 1402 defects found.
- Rubey75: the data presented in this paper were obtained from validation efforts, generally involving relatively small real-time control programs (averaging about 32 K machine-language instructions).
- Shen85: this study was based on the detailed analysis of three products developed at IBM's Santa Teresa Laboratory. The products studied were developed and released since 1980. Product A is a software metrics counting tool written primarily in Pascal of total size is 7 K TSI (Total number of source instructions in thousands, excluding comments). Product B is a compiler written primarily in PL/S (a derivate of PL/1), of size 94 K TSI. Product C is a database system written primarily in Assembly Language, the total size is 326 K TSI.
- Withrow90: the object of this study was the Ada software for the command and control of a military communications system. The project is composed of 362 Ada Packages, containing 114,000 lines of code. The average package was 316 lines.

This brought the total number of papers used in the analysis to 18 (representing the work of 26 authors). From these papers, 332 plain text statements of potential hypotheses were found during the extraction step (as described in Section 4.1). During the integration step (Section 4.2), 141 of these statements were found to be at the level of belief/opinion only, and thus were excluded from further analysis; the remaining 191 plain text statements were formalized, resulting in 396 specific hypotheses. These metrics describe the hypothesis base as it was found at the beginning of the analysis.

During the analysis step (Section 4.3), visualization and data mining were applied on the formalized hypotheses, which were then traced back to the plain text hypotheses from the various sources. Using this analysis, 80 of the 191 plain text hypotheses were found to be related to the topics highlighted by Marick as the common points found in his analysis (The remaining plain text hypotheses dealt with a wide range of other topics, such as maintainability, effort, reliability, reparability, memory constraints, reusability, and cohesion, which are outside the scope of our current analysis. For the purpose of comparability, we chose to analyze only the intersection of our hypothesis base with topics identified by Marick, rather than aim for completeness).

As a result of this analysis, we found almost the same results as in the Marick study, which used a manual approach to abstracting information based on empirical evidence. Table 1 compares the results produced from the two methodologies

As a result of this comparison, we can see that the formal methodology had the advantage of making the links between the underlying data and the conclusions more explicit (e.g. conclusion number 7). In other cases, the conclusions were made more robust because more sources and more specific hypotheses resulting from them could be explicitly associated (e.g. conclusions number 1 and 2).

Through the comparison to the less formal approach, we did find a flaw in our methodology, namely that we were not analyzing and extracting information from figures. That is, information which happened to be encoded as figures or graphics, and which could be extracted from the paper, had not

Table 1

No.	Conclusion	Support listed in Marick analysis	Support found in hypothesis base
1	Defects in programming logic (path selection) are common	[8,11,16,23,28]	All of the same, plus: [3,22] (11 specific hypotheses)
2	Defects of omission are important	[3,11,22]	All of the same, plus [6].(Also [18,27] which were not in Marick's input set.) (29 specific hypotheses)
3	Data handling is more defect-prone than computation	[3,11,16,22,23,28]	All of the same (eight specific hypotheses).
4	Modified modules are no more defect-prone than new modules	[3] and additional sources that are not clearly referenced	Support from: [3,27,28] (3 specific hypotheses)
5	There is evidence that both small and large modules are more defect-prone than medium-sized modules	[3,28,33]	All of the same plus [9,23]. (Also [26,27] which were not in Marick's input set.) (10 specific hypotheses)
6	In abstract descriptions (requirements/functional specifications), beware of incorrectness, omissions and inconsistencies, in that order.	[2,6]	All of the same plus [3,28,32].(13 specific hypotheses)
7	Bug fixes cause a small number of new bugs	Unclear	[3,22,23,32] (6 specific hypotheses)

made it into our hypothesis base, resulting in missing support for some conclusions that Marick had found. As a result, we decided to insert this step into the extraction of information from papers. Rerunning the methodology using this step resulted in the data as show in Table 1.

We could see in this case study that we can replicate results from other manual reports, but with an advantage of that the review was conducted according to explicit and reproducible methodology.

6. Summary

In order to provide useful, not to mention accurate, decision support about software development practices and their effects on projects, we need not only empirical studies, but empirical studies that cover a range of interesting development environments. Due to the wide range of influencing factors, and the fact that we cannot yet confidently specify them all ahead of time, we need a large dataset from which observations about influencing factors can be built bottom–up. Building such an adequate dataset would be impossible if we cannot make use of existing data, even if it was never designed to contribute to a larger empirical base.

We have demonstrated two processes with which we have been working to construct such an empirical base: one is a manual and informal approach, which relies on identifying common beliefs or 'folklore' to identify useful hypotheses and a manual analysis of the information in papers to investigate whether there is support for those hypotheses; the other is a formal approach based around encoding the information in papers into a structured hypothesis base that can then be searched to organize hypotheses and their associated support. There are some advantages of having a structured base, even when it does not produce many new results in comparison to the more informal analysis approach. The advantages are:

(a) The structured base is reusable, in the sense that it can be reused to test additional research questions that may not have been known at the time of its creation.
(b) If another researcher wants to validate the results, the process is repeatable.
(c) The analysis process can be undertaken by researchers who do not have in-depth knowledge about the context and details of the individual studies. The structured information helps the researcher to see the important information to do the analysis.

Having looked at a collection of datasets and abstracted up a set of conclusions on specific topics, we have shown that the formal methodology can produce useful and feasible results, especially when it is compared to the results output from the more manual, expert-based approach. The formalized approach, by relying on a reusable hypothesis base, is repeatable and also capable of producing a more thorough basis of support for hypotheses, including results from papers or articles that may have been overlooked or not considered by the experts.

In terms of its contribution to supporting the building of knowledge across studies, we feel that the work described in this paper demonstrates that:

- There is value in multiple studies for both supporting and not supporting hypotheses. There are several instances above where the conclusions from multiple datasets all point in the same direction, thus making the overall conclusion much stronger than if it came from any single study in isolation. And, in several important instances, the results from additional studies identify important caveats by examining processes in new environments.
- Care must be taken to make sure that the objects of the comparison are actually like things that can support the conclusions being drawn.

In short, there are insights to be gained from the collection and analysis of defects according to different classification schemes, independent of the scheme. Our results show that interesting abstractions can be drawn by comparing defect information opportunistically, based on points of similarity where they occur.

Acknowledgements

This work is partially sponsored by NSF grant CCF0438933, 'Flexible High Quality Design for Software'.

References

[1] Sima Asgari, Lorin Hochstein, Victor Basili, Jeff Carver, Jeff Hollingsworth, Forrest Shull, Generating Testable Hypotheses from Tacit Knowledge for High Productivity Computing, In: Proceedings of the Workshop on Software Engineering and High Performance Computing Applications (held at ICSE 2005), St Louis, MO, USA.
[2] Basili, R. Victor, Weiss, David, Evaluation of requirements document by analysis of change date, Proceedings of the Fifth International Conference on Software Engineering, 1981, pp. 314–323.
[3] V.R. Basili, B. Perricone, Software errors and complexity: An empirical investigation, Communication of the ACM 27 (1) (1984) 42–52.
[4] V.R. Basili, G. Caldiera, D.H. Rombach, The experience factory, Encyclopedia of Software Engineering, 2, 1994, pp. 469–476.
[5] V.R. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, IEEE Transactions on Software Engineering 25 (4) (1999) 456–473.
[6] T.E. Bell, T.A. Thayer, Software requirements: are they really a problem? Proceedings of the 2nd International Conference on Software Engineering, IEEE Press, 1976, pp. 61–68.
[7] A. Brooks, J. Daly, J. Miller, M. Roper, M. Wood, Replication of experimental results in software engineering, Technical Report ISERN-96-10, Department of Computer Science, University of Strathclyde, Glasgow, 1996.
[8] A. Dniestrowski, J.M. Guillaume, R. Mortier, Software engineering in avionics applications, Proceedings of the 3rd International Conference on Software Engineering, IEEE Press, 1978, pp. 124–131.
[9] A. Endres, An analysis of errors and their causes in system programs, Proceedings of the International Conference on Reliable Software, Los Angeles, CA, 1975, pp. 327–336.
[10] L.R. Gay, Educational Research: Competencies for Analysis and Application, fifth ed., Prentice-Hall, 1996, ISBN: 0-02-340814-6, pp. 61–66, 72–73.

[11] L. Glass Robert, Persistent software errors, IEEE Transactions on Software Engineering (1981).
[12] IEEE. Software Engineering Standards. IEEE Computer Society Press, 1987.
[13] D. Keim, Information visualization and visual data mining, IEEE Transactions on Visualization and Computer Graphics 8 (1) (2002) 1–8.
[14] A. Barbara Kitchenham, Tore Dybå, Magne Jørgensen, Evidence-based software engineering, Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, 23–28 May, IEEE Computer Society Press, 2004, pp. 273–281.
[15] M. Klein, Combining and relating ontologies: an analysis of problems and solutions, in: A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt, M. Uschold (Eds.), Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA, Aug. 4–5, 2003.
[16] Myron Lipow, A. Thomas Thayer, Prediction of software failures, Journal of Systems and Software 19 (7) (1979) 71–75.
[17] Marick Brian, A survey of software fault surveys, Technical Report UIUCDCS-R-90-1651, University of Illinois, 1990.
[18] Y. Mashiko, V.R. Basili, Using the GQM paradigm to investigate influential factors for software process improvement, Journal of Systems and Software 36 (1) (1997) 17–32.
[19] J. Miller, Replicating software engineering experiments: a poisoned chalice or the holy grail, IST 47 (4) (2005) 204–233.
[20] F. Natasha Noy, Heiner Stuckenschmidt, Ontology alignment: an annotated bibliography, Semantic Interoperability and Integration, 2005.
[21] M.C.F. Oliveira, Haim Levkowitz, From visualization to visual data mining: a survey, IEEE Transactions on Visualization and Computer Graphics 9 (3) (2003) 378–394.
[22] J. Ostrand Thomas, J. Weyuker Elaine, Collecting and categorizing software error data in an industrial environment, Journal of Systems and Software 4 (1984) 289–300.
[23] D. Potier, J.L. Albin, R. Ferreol, A. Bilodeau, Experiments with computer software complexity and reliability, Proceedings of the 6th International Conference on Software Engineering, IEEE Press, 1982, pp. 94–103.
[24] H.D. Rombach, V.R. Basili, A quantitative assessment of software maintenance: an industrial case study, Conference on Software Maintenance, Austin, Texas, 1987.
[25] J. Raymond Rubey, Quantitative aspects of software validation. Proceedings of the 1975 International Conference on Reliable Software, in SIGPLAN Notices, vol. 10, No. 6, pp. 246–251, June 1975.
[26] W. Richard Selby, R. Victor Basili, Error localization during software maintenance: generating hierarchical system descriptions from the source code alone, Proceedings of the Conference on Software Maintenance, Phoenix, AZ, 1988.
[27] R.W. Selby, V.R. Basili, Analyzing error prone system structure, IEEE Transactions on Software Engineering (1991) 141–152.
[28] V.Y. Shen, T.J. Yu, S.M. Thebaut, L.R. Paulsen, Identifying error-prone software—an empirical study, IEEE Transactions on Software Engineering 11 (4) (1985) 317–324.
[29] B. Shneiderman, B. Johnson, Treemaps: a space-filling approach to the visualization of hierachical information structures, Proceedings of the IEEE Information Visualization, 1991, pp. 275–282.
[30] B. Shneiderman, Tree visualization with tree-maps: 2-d space-filling approach, ACM Transactions on Graphics 11 (1) (1992) 92–99.
[31] Visser, R.S. Pepijn, Jones, M. Dean, T.J.M. Bench-Capon, M.J. R. Shave, An analysis of ontological mismatches: heterogeneity versus interoperability, AAAI 1997 Spring Symposium on Ontological Engineering, Stanford, USA 1997.
[32] D.M. Weiss, V.R. Basili, Evaluating software development by analysis of changes: the data from the software engineering laboratory, IEEE Transactions on Software Engineering (1985) 157–168.
[33] C. Withrow, Error density and size in ada software, IEEE Software (1990) 26–30.