

Process Evolution: a Toolkit and Example for OO Design Inspections

Forrest Shull

Fraunhofer Center - Maryland
University of Maryland
4321 Hartwick Road, Suite 500
College Park MD 20742
301-403-2705
fshull@fc-md.umd.edu

Jeffrey Carver

Experimental Software Engineering Group
Department of Computer Science
A.V. Williams Bldg.
University of Maryland, College Park
College Park MD 20742
301-405-2721
carver@cs.umd.edu

Guilherme H. Travassos

Computer Science and System Engineering
Department
COPPE
Federal University of Rio de Janeiro
C.P. 68511 - Ilha do Fundão
Rio de Janeiro – RJ – 21945-180, Brazil
ght@cos.ufrj.br

ABSTRACT

Effective software development processes – that is, procedural guidelines for accomplishing specific development tasks, such as inspections – are not static, one-size-fits-all entities. Processes often do not yield promising results simply by being used “as is” on a real industrial project with budget and time constraints. Rather, effective processes are the result of evolution over time; perhaps starting with proof-of-concept studies that direct further work, being tailored to the needs of a particular organization, or incorporating an organization’s already effective work practices. Typically, such evolution relies on empirical studies to discover what types of processes are useful in practice, not in theory or opinion.

In this paper, we present a “toolkit” for process evolution based on our experience, describing types of studies and data collection methods and the situations to which they are best suited. We illustrate process evolution with a detailed example of how we evolved a complete process for inspecting Object-Oriented designs.

Keywords

OO design inspections, software process, software quality, empirical studies

1 PROCESS EVOLUTION

The argument for defining processes for specific software development tasks should be a familiar one. A well-defined process can be observed and measured, and thus improved. Processes can be used to capture the best practices for dealing with a given problem. The adoption of processes also allows for dissemination of effective work practices to occur more quickly than the building up of personal experience. An emphasis on process helps software development become more like engineering, with

predictable time and effort constraints, and less like art. [11]

(By “process” we are referring to a set of procedural guidelines for accomplishing specific development tasks, such as inspections. The type of process that we discuss in this paper will fit inside of development structures such as the waterfall lifecycle or frameworks like the CMM.)

Because each development environment is unique, there is no such thing as a *one size fits all* process. Processes must be tailored to fit the local environment. This tailoring can take different forms. There may be global or high level process issues that need to be changed before even evaluating the process, such as the number of inspectors or the design notation to be used in the inspection. An organization’s existing best practices may be incorporated. Or, the organization may be uncertain of the benefits to be gained from specific parts of the processes and choose to evaluate those parts more intensively.

In any case, the task of process improvement is an ongoing one. A process cannot be tailored only once and then left alone. The tailoring that was done must be evaluated to determine if it was helpful or not. Also, because the process is now described in a slightly different way, new ways to tailor may become evident that were not seen with the initial process. Because the goal of any process is to improve software development, continual evaluation and improvement are necessary. A process that is imposed rather than tailored and evaluated may enable an organization to say that a process is in place but often does not contribute to improved software development.

The Quality Improvement Paradigm (QIP) [1] has been recommended as a model for continual process improvement. The QIP is a set of six steps, based on the scientific method, that are executed in a repeated cycle. More specifically, the first step in the QIP is to characterize current practices (observation). Secondly, the organization must set goals. With these goals in mind the third step is to decide on changes that are presumed to improve the process (hypothesis forming). Fourth, the process must be executed. Fifth, the results of the execution must be analyzed (analysis of results and hypothesis testing).

Finally, the lessons learned must be packaged in a way that is useful at a later time. This result should lead to a refined hypothesis, with which the cycle can begin again.

The QIP provides an improvement framework but does not describe how exactly to study the process on any given iteration. The rest of this paper addresses just this issue. On each cycle of the QIP, a goal needs to be chosen, largely based on the results of the previous cycle. Each time through, the process becomes better understood and the improvements can be more effective.

Because of the already tight schedules and lack of flexibility on most industrial software projects, it is necessary to do some preliminary validation of a new process before attempting to use it on a real project in industry. Even processes for which effectiveness has been demonstrated need some initial validation to show their applicability in a new environment. Therefore, each time the QIP is applied a goal must be chosen suited to the maturity of the process, with multiple iterations through the QIP before risking the success of a real project. A suggested series of goals for a new process is as follows, with each goal establishing a certain level of confidence before more effort is expended on tailoring:

- Is this new process feasible? Are there high-level issues that must be addressed?
- Although the process is feasible, should individual steps be tailored to better fit the environment?
- Can this tailored process be used inside of a real development process, in a controlled fashion?
- Can this tailored process be used inside the less controlled environment of a real project in industry?

Based on the results desired from each of these goals, the types of analysis that will be done has to be determined. Based on the goals and analysis, the process can then be defined and executed.

Section 2 presents a toolkit containing a set of evaluation methods that we have found effective. Each method addresses a particular goal, helps the process engineer evaluate a particular aspect of the process, and allows for the continual evolution of the process. Section 3 contains a brief introduction to a specific process, Object Oriented Design Reading. Section 4 discusses how the tools from the toolkit have been used to evolve that reading process. Finally, section 5 presents our conclusions.

2 A TOOLKIT OF STUDIES FOR PROCESS EVOLUTION

Many different types of empirical studies exist that are useful for work in software engineering. However, our experience has been that certain types of studies and strategies for data collection are more relevant to process work than others, and even those can be more or less

suitable for different stages of process evolution. In this section we present a “toolkit” in which we list the methods that have been most useful in our own experience, and indicate where and how they are most applicable.

One important distinction to make first is regarding qualitative versus quantitative data. Quantitative (numerical) data is useful for measuring a particular aspect of a process, such as “number of defects detected”, while qualitative data (expressed in words) is useful for getting a much richer understanding. Both types of data are necessary to evolve processes in useful ways. Quantifying the effects of a process is useful to support decision-making, but much useful insight will also come, in qualitative form, from the subjects executing the process in ways that are not easily reducible to quantitative data. Most of the tools in this toolkit are thus involved in collecting some mixture of the two data types.

It is also important to call attention to the two types of lessons learned from empirical studies: *global* issues that affect the entire process, versus *specific* issues that affect individual process steps. While the distinction seems obvious, it is important to note because individual studies tend to concentrate on one or the other, either validating the overall focus and direction of a process or fine-tuning the individual steps to increase effectiveness.

We group our experience reports into the categories of study types (how to organize data collection and analysis to support investigation of hypotheses) and data collection (how to get the relevant information about a process).

2.1 Types of studies

The “type of study” is basically a framework that organizes the use of the process in some environment, the collection of data about the process, and the analysis of the data. Different types of studies are more appropriate for different environments and different points in process evolution.

Case studies examine a particular process in the context of a larger software lifecycle. They are thus the best way to provide an answer as to whether a process can be useful to an organization under real development conditions. However, case studies are not suitable vehicles for understanding a completely new process. Case studies are expensive – developers must be trained and must overcome the learning curve, and their time is expensive – and it can be dangerous to try out untested processes (as with any other new technology) on projects with real time and budget constraints. Previous empirical studies can reduce the risk of a new process.

Case studies can incorporate different levels of rigor, ranging from more controlled studies (looking at a real lifecycle, but using controls such as a replicated or “baseline” project to study particular variables) to more realistic (done in industry with real time and budget pressures and professional developers, as well as many

uncontrolled factors that can potentially influence events. More about case studies can be found in an overview paper by Kitchenham *et al* [7].

Controlled, quantitative studies are the most rigorous way to collect data as to a process' overall effectiveness, but expensive to perform and thus not always practical as a way to evaluate a new process. To address this problem we often perform **feasibility studies** (sometimes referred to as "quasi-experimental designs" [4]) in which data is collected according to some experimental design, although full control over all possible variables is not achieved. That is, these are studies that attempt to test the effectiveness of a process although rival hypotheses will still exist at the end of the study (e.g. we may observe changes in subject effectiveness but can't completely rule out the possibility that they were caused by something other than the process being used).

For example, a feasibility study of OO design inspection techniques [16] had no control group. Thus, we could not get a good idea of how much better than ad hoc practices the techniques were, but could evaluate whether they could be used at all, and what kinds of defects they turned up, in order to decide whether it would be worthwhile to continue investigating.

Although not conclusive, feasibility studies do provide some evidence about a process' effectiveness, well-suited for a "proof-of-concept" - some indication of whether or not it is promising enough to warrant further resources spent on its investigation. To do this, rival hypotheses must be addressed. For example, qualitative evidence may be collected as to whether the subjects themselves feel that their effectiveness improved due to the process used or for some other reason. Or, multiple studies may be run such that no one study gives a definitive answer but each study attempts to rule out a different set of rival hypotheses. In either case, the objective is not to find a definitive answer but to build up a body of knowledge that addresses the plausibility of the process' effectiveness [3].

Classroom environments are well-suited to feasibility studies. Although their results cannot be applied directly to industrial developers, running studies in the classroom allows new concepts to be tested before using them with expensive developers from industry. And, given that the population of undergraduate and graduate classrooms is including more and more industrial experience already, one of the major threats to applicability of results is diminishing in importance.

2.2. Data Collection

Typically, data is collected to determine a process' effectiveness and feasibility in a particular environment. These are, of course, separate questions; a process can be shown to be effective for a particular goal but not integrate well with a company's existing process, for example

requiring a new notation that developers are not experienced with or more effort than is feasible. Data collection should always be focused on useful measures of effectiveness (which are usually quantitative) but should include measures of feasibility or fit to the environment (which may be qualitative) whenever possible.

In some way, data collection must address the question of process conformance. Empirical results are not of much use if the researcher cannot be sure of which process produced them! Some strategies for addressing this issue are found in [3].

Data collection should always include the past experience of the subjects applying the process. Our own experiences have shown this to be one of the most important factors influencing the effectiveness of a process. Subjects almost always react differently to a process based on their past experience.

And, data analysis has to be very sensitive to potential threats to validity. That is, although not all external factors can be controlled, a well-designed experiment will collect as much data as possible on them in order to reason about their impact. Such threats cannot always be eliminated - but they do have to be enumerated and taken into account during analysis.

Strategies for data collection that we have found useful include the following (arranged roughly in order of the level of researcher effort required):

2.2.1. Analysis of created artifacts

This form of data collection results from the analysis of artifacts created during the execution of the process. This type of data can be collected with no intrusion on developers at all, if the artifacts are created during execution of the process anyway, or with minimal intrusion, if the process is changed to record additional information that will give some insight into the questions of interest.

We have found this approach most useful for quantitative measures of process effectiveness (e.g., number of defects reported, number of key functionalities achieved in a system). Our experience applying it to process conformance (e.g. analyzing use cases and test plans to judge subjects' adherence to the notation and the level of detail they used) has been that we are unable to achieve a useful level of granularity.

Artifact analysis is suitable for any phase of process evolution, and in fact, is necessary for almost any iteration.

2.2.2. Questionnaires/Surveys

Questionnaires and surveys, as retrospective methods, are useful ways to collect both qualitative and quantitative data. Although they do have some drawbacks that make them not suitable for collecting a finely-grained level of detail, there are advantages, primarily that they are not

time-consuming for the experimenter. While designing effective questionnaires does take some time, the fact that the same questions are sent out to multiple people allows the easy aggregation of answers to give a quick snapshot of responses. Even when the questions are very open-ended, and subjects are allowed to write more free-form answers, the researcher is assisted since the questions provide a framework or initial categorization for the qualitative responses.

As alluded to above, questions can range from multiple choice (requiring the least subject time to answer, but providing less insight) to long answer (in which the subject can explain his or her reasoning in more depth, but answers are harder to compare across subjects). Multiple choice questions should be designed using enough categories to provide a reasonable degree of granularity, but few enough that subjects can understand relatively easily the distinctions between them. Having a large number of choices also runs the risk that few subjects will fall into any given category, complicating the task of abstracting patterns from the data. However, categories can be designed so that they can be “collapsed” during analysis; for example, subjects might be given multiple options reflecting how much industrial experience they have with a technique, but all of these categories could be grouped into a single level, “has industrial experience,” during analysis. Long answers provide a good opportunity to get subjects’ thoughts in more detail; but complicates qualitative analysis (picking out patterns from free-form responses can be difficult). However, such qualitative analysis is the best way to gain insight into the use of the process.

The data collected can be quantitative (“75% of subjects agreed that the process was too cumbersome to use on a day-to-day basis”) or qualitative (“a majority of subjects independently identified the formality of the notation as the problem”). Questionnaires and surveys are well-suited to almost any phase of process evolution; we have found them useful for collecting experience data and background data from the subjects; getting general impressions from the subjects as to their experiences with the process; and collecting answers to specific questions we wanted answered about the process.

2.2.3. Interviews

Similar to questionnaires and surveys, interviews are retrospective methods that collect similar types of data. They are more time-consuming for the experimenter, especially when the time to schedule a suitable appointment for both parties is included. And, there are additional problems with accuracy because the subject cannot be anonymous and, consciously or not, the interviewee may want to “please” the researcher with his or her answers. However, the compensating benefits of interviews are that they can be more free-form than questionnaires and surveys, allowing the interviewee to convey information in

a way that makes sense to him or her, as well as more dynamic, allowing the investigation of topics the researcher might not have even known were important before the start of the interview.

2.2.4. Observation

We use the term “observational” to define a setting in which an experimental subject performs some task while being observed by an experimenter. The purpose of the observation is to collect data about how the particular task is accomplished. Observational techniques can be used to understand current work practices that can be incorporated into the new process. They are also useful for getting a fine-grained understanding of how a new process is applied. The observer is there to capture information about the circumstances in which the subject experiences problems or has trouble understanding the new process. The observer can also take note of the time consumed by each step of the process and whether or not the step was effective in achieving its goal.

An observational approach can be a bit more time-consuming for the experimenter and less relaxed for the subject than post-mortem interviews or questionnaires. However, we have found that the observational approach delivers more accurate qualitative results than such *retrospective* methods. When retrospective methods are used subjects may find it difficult to reconstruct their own thought processes, or may (intentionally or accidentally) present their thought processes in a more structured and coherent way than actually occurred.

Data collection in an observational study can be split into two subtypes, observational and inquisitive. *Observational data* is collected while the process is being executed, but without direction from the researcher. For instance, subjects are told to think out loud as they execute a technique. This allows the researcher to gain insight into how the process is executed, for example, the researcher can record if the subject becomes confused or does not know what to do next at any step of the process. Because the researcher should avoid interfering with the process, observational data collection is mostly passive [15].

Inquisitive data is collected at the completion of a process step, rather than during its execution. The researcher is required to be more assertive, and solicit responses to predefined questions rather than passively observe. For example, at the end of each step, the researcher could ask the subject for qualitative feedback as to whether it was worthwhile or the same results could have been better achieved in a different way. This is not information that the subject would normally think about while executing the process, yet it is invaluable to collect at this time, while it is still fresh in the mind of the subject.

A few examples of the questions that can be addressed by observational methods include:

- **Time taken (observational data):** Is extra time required because process steps are counter-intuitive or badly ordered?
- **Problems encountered (observational data):** What types of problems prevent users from applying the process as it was meant to be? Such problems can concern organizational issues, semantic issues, or the format of software artifacts being inspected.
- **Influence of prior knowledge (inquisitive data):** Does the process assume certain knowledge that users may or may not possess?

2.2.5. Others

Other useful techniques are found in [15].

An effective process evolution requires picking the right tools from the toolbox to use in combination. As a more in-depth example of how these tools can be used in combination, to evolve a process from an initial specification to a set of guidelines useful to users, we present our work on reading techniques for inspecting Object-Oriented (OO) designs.

3 INTRODUCTION TO READING TECHNIQUES FOR OO INSPECTIONS

Most publications concerning software inspections have concentrated on the organizational aspects, such as the best number of participants and the number and structure of inspection meetings (e.g. [5, 6]). The technical dimension, concerned with what techniques are actually followed by reviewers, has not been emphasized enough. That is, individual reviewers are assumed to be able to effectively detect defects in software documents on their own. As a result software reading is often done in an ad hoc or unstructured way; developers are taught how to write intermediate artifacts but rarely how to read and analyze them effectively. This is a problem because reading expertise is built up only slowly, through personal experience that may not be helpful or easy to communicate from one reader to the next. [14]

Recent research into *software reading techniques* aims to improve software reading. A Reading Technique can be defined as a series of steps for the individual analysis of a textual software product to achieve the understanding needed for a particular task. This definition has three main parts. First, the *series of steps* gives all readers a common process for achieving the goal that can later be improved based on experience. Without a standardized process, readers use their own processes, and improvement is much more difficult. Secondly, a reading technique is for *individual analysis*, meaning that the technique supports the understanding process within an individual reader. Finally, the techniques strive to give the reader the understanding needed for a *particular task*, meaning that the reading techniques have a particular goal and try to produce a certain level of understanding related to that goal [12].

The reading technique approach can be tailored to the document being inspected, the environment within which the inspection is being performed, and other aspects. This tailorability is an important attribute of reading techniques, by which we mean that each reading technique defined is specific to a given artifact and to a goal.

Previous research has shown that reading software artifacts to detect defects is worthwhile and possible. This defect detection has been shown in the reading of requirements, both natural language requirements [12] and requirements written in formal notation [9], as well as reading for usability defects [18]. We have also shown that software reading is possible in the OO paradigm, by developing techniques to read OO code and design for reuse [2].

In the work we describe here, we extended that prior knowledge by building techniques that allow OO designs to be read in order to detect defects. Because OO designs are quite different from structured designs, and increasing in popularity, new reading techniques tailored to the OO world are needed. Additionally, work in this area combined with previous research can help illuminate high-level questions in reading technique research.

An OO design is a set of diagrams concerned with the representation of real world concepts in the problem domain, but built at a different time, using a different viewpoint and abstraction level, than the requirements. When high-level design activities are finished, the diagrams can be inspected to verify whether they are consistent among themselves, *horizontal reading*, and if the requirements were correctly and completely captured, *vertical reading* [16, 17].

Ensuring the quality of the high-level design has benefits for software quality. First, by focusing the techniques on the high-level design, we are ensuring that developers understand the problem fully before trying to define the solution, which will appear in the low-level design. Secondly, it is important to locate and remove as many defects as possible at the higher level, because they become more difficult and more expensive to fix if they are allowed to filter down into the low-level design, or even the code [8, 10].

4 EVOLUTION OF THE TECHNIQUES

The evolution of the OORTs was supported by a series of empirical studies since 1998. The sequence of studies and evolution of goals are illustrated in Figure 1.

4.1. Is the idea sound?

Initial validation was accomplished by means of a study [16] that evaluated the feasibility of applying reading techniques to an OO design. Based on lessons learned from studying requirements inspections, and different types of OO design defects, an initial set of techniques was created. We chose to run a feasibility study, with feedback on form and content as a secondary goal, before expending

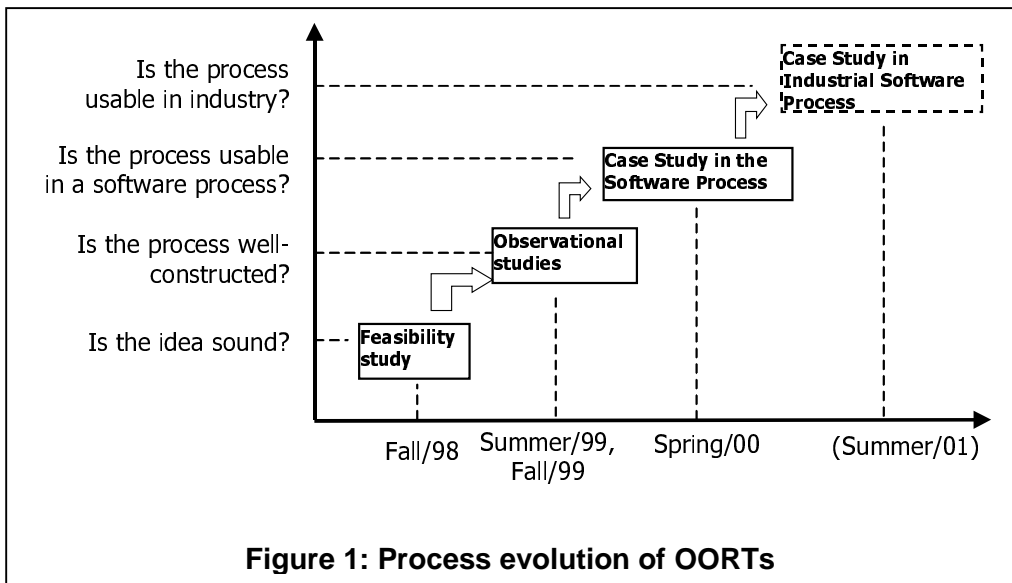


Figure 1: Process evolution of OORTs

effort to perfect the techniques.

Subjects: The subjects in this experiment came from a senior-level undergraduate software engineering course at UMCP during the Fall 1998 semester. Of the 44 students in the class, 32% had some previous industry experience in software design while 59% had classroom experience with design but had not used it on a real system (9% had no prior experience at all in software design). All students were trained in OO development, UML and OO software design activities as a part of the course. The subjects were randomly assigned into 15 teams (14 teams with 3 students each and 1 team with 2 students) for the experiment.

Materials: The materials under study during this experiment consisted of the initial version of the reading techniques. They were applied to the design of a “Loan Arranger” system responsible for organizing the loans held by a financial consolidating organization, and for bundling them for resale to investors. It was a small system (11 classes in high-level design), but contained some design complexity due to non-functional performance requirements.

Procedure: Previous to the study, subjects performed an inspection of the requirements for the system, to better acquaint themselves with the given system and the domain. Subjects were then given the “corrected” requirements (based on the aggregate inspection results of the class) and use cases and asked to design the system. The “best” design, as chosen by course instructors, was distributed to the class and subjects were asked to perform a design inspection of it.

We should note here that we had no control group; that is, we could not compare the OORTs’ effectiveness to that of another OO inspection method. There were two reasons for this decision. The first is that we are aware of no other published methods for reading OO designs. Secondly, in a

classroom environment, it was not possible to provide instruction on a topic to only a portion of the class.

Each team applied all seven of the reading techniques, but the techniques were divided up to reduce workload: One member performed the vertical reading, while the other two divided the horizontal techniques between them. After performing their individual reviews, the team members met to compile their individual defect lists into a final list that reflected the group consensus.

Data Collection: To evaluate the feasibility of the techniques, we used three of the data collection techniques from section 2.2: Questionnaires and interviews were used to collect qualitative data that addressed the question of feasibility directly, while analysis of artifacts was used as a control on the data quality and process conformance. Using both questionnaires and interviews allowed us to collect qualitative data at different times, under different conditions; evaluating the consistency of answers provided a first level of a check on data quality. The qualitative data concerned:

- Opinion of effectiveness of technique (measured by what percentage of the defects in the document subjects thought they had found)
- Subjective usefulness of different perspectives (open-ended question)
- How closely subjects followed technique (collected qualitatively and quantitatively, for consistency)
- Were the techniques practical, would subjects use some or all of them again (open-ended question)

The questionnaires were also used to capture limited quantitative data, namely the time required for individual review.

Analysis of the subjects’ defect lists yielded quantitative data concerning the number and type of defects detected by the techniques. (Because this was mainly a feasibility study, we made the assumption in our counting of defects that all defects reported were real problems with the document.)

Results and Lessons Learned: The quantitative data from this experiment showed some positive results. Using the techniques did allow teams to detect defects (11 were

reported, on average), and in general subjects tended to agree that the techniques were helpful. Also, the vertical techniques tended to find more defects of omitted and incorrect functionality, while the horizontal techniques tended to find more defects of ambiguities and inconsistencies between design documents, lending some credence to the idea that the distinction between horizontal and vertical techniques is real and useful. Thus, the data supported the conclusion that the techniques were feasible: they could really be used to detect defects, and moreover could be used to target particular types of defects.

At the same time, the qualitative data also indicated that the techniques were not as well-specified as they could be. From the qualitative data we were able to learn three global lessons on how to improve the techniques for the second version. The first lesson that we learned was that **OO reading techniques should concentrate on semantic, not syntactic, issues**. Subjects found syntactic checking tedious but saw more use in the steps trying to validate whether design decisions make sense and are feasible, based on the reviewer's knowledge. The second lesson learned was that **reading techniques need to include not only instructions for the reader, but some motivation as to why those instructions are necessary**. Subjects tended to lose sight of the larger goals if they were not well-articulated, possibly performing sub-tasks erroneously because they did not see how the information yielded fit into the larger picture. The third lesson that was learned was that **the level of granularity of the instructions needs to be precisely described**. Discussing functionality is a difficult but necessary part of the reading techniques. The difficulty comes from the many different levels of granularity at which system behavior can be described, and just assuming that subjects will intuitively grasp the correct level of granularity is naïve and causes frustration for the reviewer.

These results led us to produce a second version of the techniques that incorporated several global changes, such as a greater focus on semantic checking, more explanation of the goals of the process steps, and a new terminology to help discuss system functionality in more detail.

4.2. Are the techniques well-constructed?

The new version of the techniques was then studied using another feasibility study, since we were still interested in studying the techniques in isolation, rather than applied as part of a full software lifecycle. The reason for this was primarily that we wanted some indication about the problem domains, and the background of inspectors, for which the techniques could be most useful. The risk of introducing the techniques on an unsuitable project, with time and budget constraints, when their ease of use had not been tested also implied that another feasibility study could be useful.

To get the level of detail about the techniques that we wanted, we used an observational approach (i.e., using

experimental methods suitable for understanding the process by which subjects apply the techniques) [16]. Because this observational approach was a somewhat unusual approach, we first performed a pilot study to debug the observational approach and get it to work in our setting. Only after that did we perform a full-scale observational study, reported below. The observational approach was necessary to understand what improvements might be necessary at the level of individual steps, for example, whether subjects experience difficulties or misunderstandings while applying the technique (and how these problems may be corrected), whether each step of the technique contributes to achieving the overall goal, and whether the steps of the technique should be reordered to better correspond to subjects' own working styles.

Subjects: The 28 subjects were members of a graduate-level Software Engineering class at the University of Maryland during the Fall 1999 semester. Of the 14 that actually performed the OO inspection, 86% had previous industry experience with OO design and the other 14% had classroom experience. The subjects were grouped into two-person teams in order to carry out the experiment. One member of the team acted as the executor (responsible for applying the procedure) and the other member was the observer (responsible for recording observations about how the procedure was executed). All students received training on the OO reading techniques and the observation process.

Materials: The materials under study during this experiment consisted of a new version of the OO reading techniques. They were applied to two designs: one for the Loan Arranger (LA) system, described in the last section, and one for an automated parking garage control system (PGCS). The Loan Arranger design used in this study was a simpler version of the same system described in the last section (7 classes in the high level design, 4 interaction diagrams and 3 state diagrams). The PGCS was responsible for allowing drivers to enter and leave a parking garage and keeping track of monthly parking tickets as well as the number of available spaces for general parking. The PGCS was a relatively small system (6 classes in the high level design, 5 interaction diagrams and 2 state diagrams). The LA problem domain was selected due to its unfamiliarity to reviewers, while the PGCS domain was familiar.

Procedure: A quasi-experimental, factorial design was used in which half of the class reviewed the LA design, and the other half the PGCS. In each of these groups, roughly half the teams had previously inspected the requirements document for the same system. In this scheme, we could look for any differences in performance due to the reviewers' past familiarity with the system requirements or with the problem domain.

Before the study, subjects received training in the reading techniques to be applied and the observational methods. Training in observational methods was accomplished by

presenting the roles of executor and observer and defining their specific responsibilities. One member of each team was assigned to be the executor and the other to be the observer. Subjects were asked to come up with their own questions for eliciting observational and inquisitive data. After the execution of the techniques, each team wrote an evaluation report discussing their experience and the results of the observation.

Data Collection: This study used two of the data collection techniques from section 2.2: analysis of artifacts and observation. Analysis of artifacts was again used for collecting some quantitative data, namely the time required for executing the techniques and the number and type of defects detected. However, observational techniques were the most important method used in this study. A rich array of qualitative data was collected through their use. As mentioned earlier, the teams produced an evaluation report, which included both a summary of the notes taken during observation as well as retrospective data determined after the execution of the process. Some of the metrics collected from the observations include:

- Executor’s opinion of effectiveness of technique
- Problems encountered with specific steps of procedure
- How closely executors followed the techniques

The retrospective data (collected via open-ended questions) provided the following information:

- Usefulness of different perspectives
- Were the techniques practical, would they use some or all of them again
- The problems found using the techniques

As can be noticed, the retrospective data are better suited to global issues, rather than the critiquing of individual steps. Also, some of the metrics collected here were the same as in the previous feasibility study, allowing a comparison of results across the two versions of the techniques.

Results/Lessons Learned: The quantitative data from this experiment, first, verified the difference between types of defects found by horizontal and vertical techniques. Secondly, they showed that having expertise in the domain was not helpful for subjects in the design inspection. The teams that inspected the design from the low-expertise domain (LA) performed better than those that inspected the design from the high expertise domain (PGCS). Finally, they showed that being a participant in a requirements inspection for the same system did not improve a subject’s performance in the design inspection.

But, the qualitative data provided us with some potential ways of improving the techniques. First, the **order of dealing with information** must match the subjects’ own way of thinking about the problem. Subjects sometimes

wanted to reorder process steps to make them fit better with their own work practices. Some preferred to use a top-down approach, e.g. construct use-cases then find the functionality to match, whereas other preferred a bottom-up approach, e.g. identify product functions then abstract up to the use cases. Such different modes of thinking may contribute to whether or not a procedure is used, as defined, in practice. Secondly, the **amount and type of training necessary** needed to be modified. For example, the subjects stated that the training was adequate for the task, but to really be effective they needed a “trial-run” of using the techniques. This should be part of the training, and should be done during the training session so that the subjects could ask the instructors questions to ensure they had a complete understanding of how to use the techniques. Finally, **differences in design approaches could hurt design inspection**. If the inspector approaches the design with a different mindset than the original designer, he or she may report many false defects, and be blinded to some real defects.

This led us to produce a third version of the techniques, using the data from the observations about the way that readers applied the techniques. This version of the techniques also focused more on the semantics behind the design models and less on the syntax. We also changed terminology, from “defects” to “discrepancies”, reflecting the fact that inspectors and designers may have different ideas about the design. Additional improvements were made regarding training and discrepancy report forms. The details of the process evolution up to this point (along with the third version of the techniques) are presented in a technical report [13]. This technical report shows excerpts from the third version of horizontal and vertical reading techniques. These techniques can be compared with the previous ones presented in [16] to observe the evolution based on these study results.

4.3. Can the techniques be used in a software process?

Previous studies had convinced us that the techniques were feasible, in that their use could detect defects and that their individual steps and ordering seemed reasonable. However, we still had no evidence that they could be used as part of a software development project, i.e. that they did not require a prohibitive amount of effort, that what they required was available in a typical development environment, and that their effects were useful for continuing the development of a system. For this understanding, a case study was planned to evaluate the techniques inside of a software development process in a classroom environment. So, it is a more realistic study than the previous one, but maintaining more control than in an industrial setting.

Subjects: The subjects in this experiment came from a senior level undergraduate software engineering course at UMCP during the Spring 2000 semester. Of the 42 students in the class, 14% had some previous experience

with OO design in industry while 45% had classroom experience with OO design but had not used it on a real system (40% had no prior experience in OO design). All the students were trained in OO development, OO software design activities, UML, and OO design inspections as part of the course. The subjects were grouped into high, medium, and low expertise categories, and one person from each group was randomly assigned to each of the 14 3-person teams.

Material: The materials under study during this experiment consisted of an evolved version of the techniques based on the results from the previous study. They were applied in the evolution of the PGCS system, described in the previous section. The students were required to add functionality to that system that allowed customers to reserve tickets and pay bills over the Internet.

Procedure: The subjects used a waterfall development process, to create an enhanced version of an existing system. Previous to the study discussed here, the subjects had created and inspected the requirements document for the complete PGCS system. After correcting the defects found during the requirements inspection, each team created a design for the system.

Once the initial design had been created, all teams used the horizontal reading to inspect their own designs to ensure that they were consistent. They corrected any defects that they found. After the designs had been correct, the teams traded designs. Each team then performed the vertical reading techniques on a design for another team. The list of defects found by the reviewers was then returned to the authors of the design for correction.

In the overall scope of the software development process there was no control group here. This occurred for two reasons: first, the design inspection was one small part of a larger experiment, and the overall experimental design did not allow for a control group. Secondly, in a classroom environment, it was not possible to provide instruction on a topic to only a portion of the class.

Data Collection: To evaluate the effectiveness of the techniques in the development process, we used two of the data collection techniques from section 2.2: questionnaires and analysis of created artifacts. The questionnaires were used throughout the development cycle to collect both qualitative and quantitative data. The quantitative data collected include both background information, used to classify the subjects as having high, medium, or low expertise, and the amount of time taken to use the techniques, used to evaluate feasibility of use. The qualitative data collected by the questionnaires concerned:

- Opinions of the helpfulness of the techniques.
- Problems encountered using the techniques, or extra knowledge that was needed to use the techniques.

- Opinions of effectiveness of training.

Analysis of the defect lists provided quantitative data about the number and types of defects found by the teams. The data was useful in determining if the output of the reading process uncovered defects and was useful for continuing the development process.

Results/Lessons Learned: Although run in a classroom environment, the qualitative data from this, our first case study, provided us with some global lessons about the techniques and how they fit with other development processes. First, we found that subjects were able to apply the techniques inside of a lifecycle and in combination with other processes (specification, design, implementation, and testing). The techniques were useful for inspections, as they resulted in defects that were corrected to improve system quality, but in the context of system development they also turned out to have another use: The vertical techniques helped students to gain a better understanding of the system functionality and how it should be represented in the design. Also, the techniques were feasible to use during development, as the effort required was not prohibitive compared to other system tasks; the design inspections required on average 20 hours per team, or 24% of the overall effort spent on design. Outside of the training in the techniques, the subjects required no special knowledge that was not previously gained during the development of the system.

Secondly, we found the techniques to be useful for teaching OO design. Specifically, we were able to use the horizontal techniques to improve the OO training. While we expected the subjects to use the techniques to look for defects in the designs, they found that what they were instructed to look for in terms of defects gave them a good idea of things that would not appear in a quality design.

4.4. Does the technology transfer to industry?

At this point in time, having run a case study in a classroom environment, our next step is to run an industrial case study to make sure these ideas can be tailored and transferred to an industrial environment. For this we are currently seeking an industrial partner. The series of studies run to date has provided a body of evidence that, first, yielded a proof-of-concept of the usefulness of the process and second, identified a set of issues that we know will be important for tailoring this process for effective industrial use. For example, we have already observed the effects of subject training and of previous subject experience (with inspections in general, with the problem domain, and with OO concepts) and how they can influence the effectiveness of the process.

Every pilot study of a new process in an industrial environment needs a back-out plan, i.e. a way of responding with minimal disruption to the success of the project if the process turns out to be ineffective in the

environment. However, the studies run to date have formed a responsible approach for developing confidence in the process under study and for ensuring that we can tailor it to the industrial environment.

5 CONCLUSIONS

In this paper, we have outlined an approach for evolving processes, from the early concept phase to the tailoring and use of the process on an industrial project. We have illustrated how a body of evidence concerning process effectiveness can be built up, using different types of studies to study different questions of interest about a process. We believe that such an approach is helpful for a responsible interaction with industry.

For researchers, we have provided a toolkit for planning such an iterative approach to evolve and study processes. We have given some indication, based on our own experience, of heuristics for deciding what type of study and what type of data collection is best suited for a given stage of process evolution.

For practitioners, we have discussed a process for addressing an important development task, OO design inspection. We have illustrated the series of studies used to build up confidence in the process and understand the relevant variables, so that readers can understand and judge for themselves the evidence regarding the process' effectiveness. We have argued that the process can continue to be adapted, adopted, and studied in an industrial environment without undue risks to the project.

We invite interested readers to our web page, www.cs.umd.edu/projects/SoftEng/ESEG/manual/OORTs/, which contains pointers to information on the studies referenced in this paper, a particular OORT at various stages of evolution, and further references.

ACKNOWLEDGEMENTS

This work was partially supported by UMIACS and by NSF grant CCR9706151. We wish to thank Prof. Victor Basili for his support and our subjects for their hard work. Dr. Travassos also recognizes the partial support from CAPES-Brasil.

REFERENCES

1. Basili, V. R. and Caldiera, G. "Improve Software Quality by Reusing Knowledge and Experience," *Sloan Management Review* 37, 1 (Fall 1995), 55-64.
2. Basili, V. R.; Lanubile, F.; and Shull, F. Investigating Maintenance Processes in a Framework-Based Environment. In *Proc. of the International Conference on Software Maintenance* (Bethesda MD, Nov. 1998), IEEE Computer Society, 256-264.
3. Basili, V. R.; Shull, F.; and Lanubile, F. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering* 25, 4 (July 1999), 456-473.
4. Campbell, D.; and Stanley, J. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin Company, Boston 1963.
5. Fagan, M. Advances in Software Inspections, *IEEE TSE* 12, 7 (July 1986), 744-751.
6. Gilb, T.; and Graham, D. *Software Inspection*. Addison-Wesley, Reading, MA, 1993, Chapters 7-8.
7. Kitchenham, B.; Pickard, L.; and Pfleeger, S.L. Case Studies for Method and Tool Evaluation. *IEEE Software* 12, 4 (July 1995), 52-62.
8. Pfleeger, S.L. *Software Engineering: Theory and Practice*. Prentice-Hall, 1998.
9. Porter, A.; Votta Jr., L.; and Basili, V. R. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE TSE* 21, 6 (June 1995), 563-575.
10. Pressman, R. *Software Engineering: A Practitioner's Approach*, 4th ed., McGraw-Hill, 1997.
11. Rombach, D. "Fraunhofer: The German Model for Applied Research and Technology Transfer." In *Proc. ICSE'00* (Limerick, Ireland, Apr. 2000), 531-7.
12. Shull, F. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. PhD Thesis, Computer Science Dept., University of Maryland. 1998.
13. Shull, F.; Travassos, G.; Carver, J.; and Basili, V. R. *Evolving a Set of Techniques for OO Inspections*. University of Maryland Technical Report CS-TR-4070. October 1999.
14. Shull, F.; Rus, I.; and Basili, V.R. How Perspective-Based Reading Can Improve Requirements Inspections. *IEEE Computer* 33, 7 (July 2000), 73-79.
15. Singer, J.; and Lethbridge, T. Methods for Studying Maintenance Activities. In *Proc. of the Workshop for Empirical Studies of Software Maintenance* (Monterey CA, Nov. 1996), 105-110.
16. Travassos, G.; Shull, F.; Fredericks, M.; and Basili, V. R. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. In *Proc. OOPSLA'99* (Denver CO, Nov. 1999), ACM Press, 47-56.
17. Travassos, G.; Shull, F.; Carver, J.; and Basili, V. R. Reading Techniques for OO Design Inspections. In *Proc. of the 24th Annual Software Engineering Workshop* (Greenbelt MD, Dec. 1999), NASA Goddard Space Flight Center (SEL-99-002).
18. Zhang, Z.; Basili, V. R.; and Shneiderman, B. An Empirical Study of Perspective-Based Usability Inspection. In *Proc. of Human Factors and Ergonomics Society Annual Meeting* (Chicago IL, Oct. 1998).