Evaluation of Extreme Programming Design Approach: A Replicated Study

René Noël, Marcello Visconti, Gonzalo Valdés, Hernán Astudillo
Universidad Técnica Federico Santa María, Departamento de Informática
Valparaíso, Chile
{rnoel, visconti, gvaldes, hernan}@inf.utfsm.cl

Abstract
Evolutionary Design is Extreme Programming’s approach to organize software structure and its relationships, encouraging refactoring, test driven development and the simplest solution for the requirements of a single iteration, thus avoiding a big up-front design activity at the beginning of the project that can cause carrying on a huge structural complexity throughout the whole project. In order to contrast this approach with a planned or traditional design approach, an empirical evaluation of impact on software design quality and process productivity has been designed and conducted in an academic environment with toy size problems. Experimental studies planning details are presented, and two replications with different experimental designs are described. Results suggest that there are no differences in quality between both approaches, and that productivity is better when a planned design is adopted.

Keywords: Extreme Programming, Evolutionary Design, Experimental Design.

1 Introduction

Extreme Programming (XP) [1] proposes an iterative process and an evolutionary design approach that encourages the design of the simplest solution for the requirements considered in each iteration, without worrying about next iteration requirements and its design complexity. This approach allows embracing change by avoiding big up-front design activity and carrying design complexity throughout the whole project, as traditional software processes propose.

There are documented cases of development teams that resist to this approach [5][8][10][15], questioning XP design practices and their naturalness; besides, a big up-front design is an instance for identifying reusable structures that can improve productivity and product internal quality. Theoretical fundaments can be argued in favour of both design approaches, but Experimental Software Engineering offers a chance to gather empirical evidence on its impact over product quality and process productivity. This paper presents two experimental designs and their results on evaluating the impact of different design approaches over product quality and process productivity.

The design topics covered in this article are limited to the detailed design activity (class models, method’s algorithms) and not architectural design issues.

Evolutionary design approach and potential problems with respect to planned approach are presented on section 2. Section 3 presents how XP practices and processes are executed during the experimental activities, and shows the design and results of the two experimental studies. Section 4 discusses the main findings of the study considering both experimental studies, its interpretation on Evolutionary Design context and future work.

2 Design Concepts on Extreme Programming

2.1 Design Strategy

XP proposes a set of practices and a process lifecycle that allows the planning, analysis, design, coding, testing and integration of software [1]. Besides, proposes an evolutionary design approach that presents a set of practices that differ from traditional design activity. In order to set a baseline for traditional (or planned) design activity, design will be defined as a big up-front activity, where the design complexity of the whole project is evaluated, modelled and rigorously documented before coding, based on a finished Requirements Analysis activity as proposed in a Waterfall Model [14].
XP design approach proposes to embrace software structural complexity only when needed, thus simplifying the implementation of potential changes to requirements during development, and avoiding wasting effort doing a big up-front system. To do this, the simplest solution that satisfies the requirements of one iteration must be implemented, then refactor the software when necessary [1]. This approach, known as Evolutionary Design [3] is based on a subset of XP practices (Test Driven Development, Refactoring and Continuous Integration, all under the concept of Simple Design) [3] that enable developers to take design decisions during the project. These practices are Continuous Integration, Testing and Refactoring. Design decisions are oriented by the Simple Design practice.

2.2 Potential Problems

To embrace design complexity only when needed requires discipline from the developers: they have to avoid thinking about the whole project complexity, and only design a solution for a current subset of requirements. This blindness can cause problems related with two topics: developers satisfaction and reusable structure recognition.

2.2.1 Developer’s Satisfaction

Although evolutionary design approach seems to be a good choice to embrace change, there are reports of cases where its implementation has not been satisfactory. University environment case studies [8][10], and software research and development organizations [5][15], have reported that participants on XP software projects do not feel comfortable with the design approach: they insist on adding complexity not required at each specific iteration, on making detailed design documentation, and on designing in advance for all the system features, following a traditional or planned design approach. If these perceived observations are correct, they could impact product design quality, process productivity, and developer’s satisfaction.

2.2.2 Recognition of Reusable Structures

The lack of perspective of the whole software design can have an impact on developers’ capability to identify reusable structures such as architectural or design patterns [4] that encapsulate software design knowledge and offer a quick and quality-guaranteed solution to recurrent design problems. Ignoring these solutions can impact on process productivity and product design quality.

2.3 Other Experimental Studies on XP and Software Design.

Experimental efforts to assess the impact of design approaches have not been reported, although experimentation on XP projects is a rising topic: [17] presents a controlled case study approach for agile software development empirical evaluation, a framework to gather experimental quantitative and qualitative data. In [16] a proposed practice and tool for improving testing practice of XP is empirically evaluated, through a controlled experiment; the perception of students about a subset of XP practices has been evaluated in [9] this perception is favourable to XP, unfortunately the study does not address design practices explicitly. Other empirical evaluations related to software design are presented in [18], where the impact of product design over maintainability is assessed. In [21] an experimental design is proposed in order to qualitatively evaluate design decisions throughout the development process, its related work section discusses studies about how personal factors of participants impact on software design. An evaluation of design specifications produced by senior-level year students is presented in [2].

3 Experimental Designs for Design Approach Evaluation of XP Projects

As mentioned in section 2.1, XP’s design approach looks for embracing change by implementing practices to avoid big up front design. As presented in section 2.2, this approach can negatively impact process productivity and software design quality. In order to compare XP approach with a traditional or planned approach, an empirical evaluation has been designed. The goal of the study is to detect the existing differences on product quality and process productivity when developing with an XP design approach versus a planned design approach. This section presents an experimental design that looks for insights on this issue, and is presented according to the experimental process proposed by [20].
3.1 Experiment Context in XP Ideal Life Cycle

The experimental activities are part of a laboratory environment, and an online observation is performed. Participants are non professional developers (undergraduate students), solving non real problems. The study is limited in order to perform the detailed design activity using a subset of XP practices that allow performing the detailed design activity, as presented on section 2.1. Considering an XP ideal project life cycle [1], the activities on the study are placed on the Development Cycle stages:

![Figure 1. Ideal Life Cycle of an XP Project](image)

Within the Development Cycle, all the practices that involve design decisions are considered. Each of them is implemented as follows:

- **Pair Programming**: implementation agreements between the developers are instances for making design decisions. This practice is fully implemented.
- **Planning Game**: choosing the priority and requirements to be implemented in each iteration is part of the study design, and not part of subject activity on her own.
- **Test Driven Development**: participants are not familiarized with automatic unitary testing tools. However, test case design must be performed, because of its design importance on defining business validations and service’s interfaces. Unitary tests cases will be defined on Test Case Cards, specifying input values and desired output.
- **Whole Team**: In order to not introduce bias due to the client interaction with subjects, all requirements are documented. Documentation must be validated in order to be precise and fully understandable by subjects. No business questions must be answered during the experimental study.
- **Continuous Integration**: on the study, a development team is composed only by a pair of developers, programming on the same workstation, so continuous integration is guaranteed.
- **Refactoring**: due to its importance on XP design approach, this practice is implemented and emphasized during training. Also, requirements and iteration planning are designed to cause refactoring between iterations.
- **Small Releases**: the activity result is considered a small release in the context of a greater project.
- **Coding Standards**: this practice is not encouraged, because only one pair of programmers made up the development team.
- **Collective Code Ownership**: same as Coding Standards.
- **Simple Design**: subjects are trained to identify simple solutions by examples and in an exercise during training.
- **Metaphor**: a metaphor can be very helpful to understand and communicate system design, depending on the problem and the insight of developers. Metaphor is not encouraged, due to the bias that could be produced by differences of power of the metaphor on different problems.
- **40 hours at week**: does not apply.

3.2 Experimental Activity Design

The software development activity (which subjects must tackle) was designed considering the practices that must be implemented, and time constraints to execute experiment trials. Design, coding and testing of software features can’t take more than two academic time slots, so we planned a 4-hour activity and design problems that could be solved in that time box.

A process based on XP original definition [1][19][7] was defined. In addition, a variation of the original XP process was performed, in order to incorporate a planned design session at the start of each iteration.

3.2.1 Activity Description

Each trial of the experimental study consists of training the subjects on a development method, and then making them apply the method on a given problem. Training sessions consists of teaching XP main
concepts, and the specific method to be applied, plus a guided execution of the method on a sample problem. The application of the methods consists basically of developing a simple software solution, whose requirements are separated in two sets. The whole context of the problem is presented on a global system description that explains the whole functionality and all the domain elements and its relationships. The first set specifies a functionality that adds value to the client, but that is not the whole system functionality. The second set complements the first one, and covers all the desired features of the system. By giving a whole system description at the start, planned design subjects can anticipate system design complexity, probably improving its final design quality, and also improving its productivity when identifying already solved problems. If too much time is spent on making this up-front design, productivity could be negatively affected.

XP approach can presumably be more productive without a first up-front design, but separating requirements in iterations guarantees that design must be at least reviewed to implement the second set of requirements, and could probably require refactoring the first iteration code.

3.2.2 Method 1: XP Design Approach

First activity is to know the project context (based on global system description document), to have a general vision of the problem. Then, the first set of requirements is read and understood. First development activity is to write test cases to ensure that functionality is rightly implemented. Next, a short design session to solve only the current iteration requirements must be performed, drafting design diagrams with low detail, just enough to communicate ideas between pairs. Next, coding and testing must be performed. After the end of the iteration, a new set of requirements is read and understood. Test cases for the new functionality must be written. The short design session must now consider refactorings when needed, in order to implement new functionality considering Simple Design concepts (basically no logical redundancy, low coupling and modularity). After the design session, developers must code and test their solution. Source code will be collected at the end of each iteration.

3.2.3 Method 2: Planned Design Approach

First activity is to know the project context (based on global system description document), and with this description, subjects must plan the design, considering all the available information. This design must be a low detail draft, but it must focus on detecting complex class relations, types, reusable structures or design patterns that could predict a good design quality from start, and could improve productivity by identifying already solved problems (as in design patterns case). Next, the requirements for the first iteration are read and understood. Tests cases are written, and then a design session is performed, consisting of identifying which part of the initial design solves the current requirements and must be implemented. Next, developers must code and test the implementation. The second set of requirements is treated exactly like the first except for the facts that now refactorings may be considered, although it is expected that won’t be needed with a good initial design.

3.3 Problems’ Design

Problems were designed to be completely solved within the time box allocated for the activity, and to present design complexity and refactoring opportunities. All problems present a business logic that forces subjects to decide where to put the complexity: on the class design or on the algorithms of the classes’ methods. Also design patterns were introduced implicitly on the business logic. For example, Problem 1 of the second experimental design consists of building a system that can create a hierarchical structure of a document; number of levels is variable but limited, and it works as follows: the user expresses how many Chapters, how many Themes and how many sections they wants to define then the system must ask for each chapter, theme or section name, and finally show the document structure; a quick and well designed solution can be implemented using the Composite Design pattern [4].

3.4 Hypotheses and Metrics

High level null hypotheses are formulated in order to evaluate the existence of differences on product quality and process productivity using XP approach and planned design approach:

\[ H_0': \text{The use of planned design approach produces software with an equal design quality than evolutionary design approach} \]
H0’: The use of an evolutionary software design approach is as productive as a planned approach on an XP project.

In order to test these hypotheses, quality and productivity are measured by using standard metrics. For each of the metrics, a null hypothesis is formulated, for instance:

**H0’1**: Productivity measured on LOC/minute is the same using evolutionary or planned design approach

Independent variables that can affect quality and productivity are identified:

- **Design Approach**: is the factor under study, and has two levels: evolutionary approach (XP’s approach) and planned approach.
- **Participant’s Design Experience**: is an undesirable factor of influence and the experimental design will focus on minimizing its impact.
- **Problem to Solve**: the project to be considered using the design approaches can influence the results. If different problems are used on the study, the impact must be minimized or evaluate if the impact on results is statistically significant.
- **XP Knowledge**: in order to minimize the influence of this undesirable factor, participants are trained on XP’s practices an process, and a guided exercise applying them is performed. This will help also on guaranteeing process conformance during the experimental study.

Business logic complexity can be addressed during the detailed design by creating an appropriate structure of classes, methods and interfaces that provides a maintainable and flexible solution, or assigning complexity to a few and highly complex methods, attempting to maintainability and understandability of the code. In order to evaluate software quality, methods’ algorithms complexity is measured.

### Process Productivity:

- LOC/minute (PTLOC)
- Number of Classes/hour (PNOC)
- Number of Methods/minute (PNOM)

### Product Internal Quality (Design Quality):

- Decision Count (DC)
- Maximum McCabe Cyclomatic Complexity (MCC), of the more complex method coded by each subject.
- Number of Code Statements (NSTMNT)

Productivity and Quality metrics are measured using tool Metrics 1.3.6, an Eclipse plug-in that implements the measurement definitions given in (Henderson-Sellers, 1996).

#### 3.5 Technical Details and Instrumentation

Instruments developed for the experimental study execution and its previous activities were Previous Surveys, Training Slides, Process Handout, Requirements Documents, Tests Case Cards (used to specify test cases for unitary testing) and Final Surveys. Development Tools used were Eclipse SDK 3.2, Java 2 SDK 1.4.2_04. For data collection, the tool Metrics 1.3.6 for Eclipse was used.

**3.6 Original Experimental Design**

The original experimental design was presented in [13], consisting on a random blocked design, with one factor (the design approach) and two levels (XP and planed approach). Participants were undergraduate students that had previously approved an Object Oriented Analysis and Design course. Two blocks were conformed by subjects with design experience and without design experience (marked with different grey levels in table 1), previously evaluated by a survey. Subjects were 31 development teams conformed by a pair of developers, with equivalent design experience. A training session of 1.5 [hours] was performed.

**Table 1: Original Experimental Design.**

<table>
<thead>
<tr>
<th>Subject</th>
<th>XP Approach</th>
<th>Planned Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Results of this experiment were presented on [12], remarkable results of acceptance test are the following:

![Figure 2: Significance Test Results for Original Experimental Design.](image)

Chart shows the t-Test 2-tailed probability of rejecting null hypothesis when it's true, on subjects blocked by experience on design experienced subjects (DE) and no experienced subjects (NDE). Significant differences on productivity for subjects with no design experience where found (p<0.07), obtaining better results with a planned design approach. Differences on software design quality are less significant, but show that experienced subjects produce higher quality software with a planned design approach. The difference between experience blocks leads us to try to mitigate the influence of this factor instead of blocking it, in order to isolate the method effect from the experience effect.

### 3.7 Second Experimental Design

To mitigate the influence of the subjects’ particular experience, each of them must use both design approaches. This implies having two distinct problems to solve, in order to not introduce maturation bias when applying both approaches on the same problem. Thus a 2x2 factorial design with repeated measures is proposed where the factors are the Development Method (with levels Method 1: XP Design Approach and Method 2: Planned Approach) and the problem to be solved. Experimental design and training activities for balancing XP knowledge and ensuring process conformance during the experimental study are presented in Table 2.

**Table 2: Second Experimental Design.**

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method 1</strong></td>
<td><strong>Method 2</strong></td>
</tr>
<tr>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>Problem 1</td>
<td>Problem 1</td>
</tr>
<tr>
<td><strong>Method 2</strong></td>
<td><strong>Method 1</strong></td>
</tr>
<tr>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>Problem 2</td>
<td>Problem 2</td>
</tr>
</tbody>
</table>

To avoid carryover effects, counterbalancing is applied: if the whole Group 1 uses Method 1 in first place and then Method 2, a bias can be introduced due to the learning of tools, programming language or others on the first session. Thus, one half of Group 1 will use method 1 and problem 1, and the other half will use method 2 and problem 2. On the second day, methods and problems will be inverted. Same strategy is applied to group 2.

Participants were 22 senior-level students, with Object Oriented Design knowledge and some professional practice. A detailed comparison of the two experimental designs is presented in [11].

### 3.8 Practical Lessons Learned and Operational Details

**Commitment of Participants:** we used incentives to get stimulated people, because volunteers are very susceptible to quit the experimentation, and forced people are susceptible to participate in a non-serious manner. A full credit grade (worth 5% of final grade app) was offered for participating.

**Preparation:** a previous survey was filled by all the potential participants. Subjects were randomly conformed, assigning two participants to a development team, conforming 11 subjects with 2 participants each one. Subjects were assigned randomly to the group 1 or 2 described on table 2.
**Flexibility of Designs:** we think volatility of available participants in an academic environment (students) calls for experimental designs that are flexible enough to accept re-assignment, and robust enough to tolerate subjects mortality.

**Re-assignment Period:** we recommend including a 15 minutes re-assignment period in the execution chronogram because of the participants’ volatility mentioned above.

**Number of Supervisors:** in order to guarantee the conformity with the process strict supervision is crucial, requiring the number of supervisors to be proportional to the number of participants. We observed that two supervisors are sufficient for a group non-greater than 40 participants; one more was required in the case of the 64 students’ course (as in the first experimental study).

**Pilot Study:** running a pilot study with a population and environment similar to those of the execution is crucial. In our case the pilot was carried out by a single person and in a different environment. The pilot allowed mitigating the risk of specifications illegibility and gave us insight about the feasibility of executing the experimental study in the stipulated time, but it didn’t allow us anticipating situations related to groups of people.

**Execution:** The first training session took 3 hours. The first trial of the experimental study was performed in 3 hours and 34 minutes, with a 15-minute break. Total considered subjects were 9, with 2 participants on each one. Second training session lasted 2 hours. There were no drop-outs. Second trial was run in 3 hours and 20 minutes, with 15-minute break. There were no new subject drop-outs. Test Case Cards, design drafts and source code were collected for each subject after the end of the activity.

### 3.9 Results Summary

Considering the drop-outs during the execution, observations of two subjects were discarded. Given the 2x2 factorial design with repeated measures, and the factors Development Method and Problem, 3 sets of hypotheses can be formulated. For Main Effect Development Method:

- **H0:** There is no difference between subjects using XP’s Design Approach and subjects using Planned Design Approach with respect to $VAR[i]$
- **H1:** There is a difference between subjects using XP’s Design Approach and subjects using Planned Design Approach with respect to $VAR[i]$

Where $i = PTLOC, PNOC, PNOM, DC, MCC, NSTMNT$. Similar hypotheses are formulated for Main Effect Problem and Interaction Effect Method X Problem. Looking for method main effect over dependent variables will allow testing hypotheses raised on section 3.4

Data was analysed applying Analysis of Variance for the 2x2 factorial with repeated measures design. Sphericity assumption was not complied, so a multivariate approach was taken. All the tests of significance for the influence of factors and interception applied (Pillay’s Trace, Wilks’ Lambda, Hotelling’s Trace and Roy’s Largest Root) lead to the same significance levels for each metric, that are presented on figure 3. Details of the analysis can be found in [11].

Looking at method’s effect, we can see that null hypotheses can’t be rejected; no differences in quality and productivity between evolutionary and planned design approaches can be demonstrated by the influence of the Method factor. Estimated marginal means for each metric for methods 1 and 2, presented on table 3 suggest that, although not statistically significant, the Planned Design Approach (Method 2) is more productive than XP Design Approach (Method 1), for every metric of productivity. Product design quality metrics are favourable to XP Design Approach for Number of Statements and Maximum Cyclomatic Complexity. Decision Count metric suggests that more control flow statements were coded when using XP Design Approach than using a Planned Design Approach.

### 3.10 Threats to Validity

A detailed analysis of threats to validity is presented in [11]. The main threats to validity are related to three key factors:

**Activity Design for Process Implementation.** Available time for perform the activity forces us to adapt XP practices and to work with toy problems, so it might be a threat for *construct validity*.

**Metrics and Problems Size.** Toy problems could be not complex enough to get significant differences on quality or productivity for the chosen metrics, attempting to *construct validity*.

**Academic Environment.** Students could not be representative of professional developers, attempting to the *external validity* of the study.
Figure 3: ANOVA Significance Test Results for 2x2 Factorial with Repeated Measures Design.

Table 3: Estimated Marginal Means for Method’s Effect

<table>
<thead>
<tr>
<th>Measure</th>
<th>Method</th>
<th>Mean</th>
<th>Std. Error</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
<td>Upper Bound</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>.458</td>
<td>.053</td>
<td>.335 .582</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>.568</td>
<td>.080</td>
<td>.385 .751</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.106</td>
<td>.162</td>
<td>.732 1.480</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.294</td>
<td>.333</td>
<td>.986 1.600</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>.079</td>
<td>.020</td>
<td>.051 .126</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>.990</td>
<td>.025</td>
<td>.933 .146</td>
</tr>
<tr>
<td>NSTMNT</td>
<td>1</td>
<td>22.167</td>
<td>.26</td>
<td>16.957 27.376</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>26.611</td>
<td>.76</td>
<td>18.043 35.179</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3.011</td>
<td>.740</td>
<td>1.995 5.317</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.056</td>
<td>.368</td>
<td>1.054 5.057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3.333</td>
<td>.717</td>
<td>1.680 4.986</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.444</td>
<td>.966</td>
<td>2.216 6.673</td>
</tr>
</tbody>
</table>

4 Conclusions

Productivity results are consistent between original and second execution of the experimental study: results suggest that a Planned Design approach always yields a better productivity. For quality metrics, in the first experimental study planned design approach yields better quality, but in the second, the results suggest that subjects using XP evolutionary approach get better quality products. However, both quality results are far from being statistically significant, so this suggests that no product design quality differences exists when using distinct design approaches.

When facing the design activity, we can choose between a planned approach or an evolutionary approach. Our study suggests that no significant differences on quality between both approaches can be demonstrated, and that process productivity is better with a planned approach, so we can evaluate the trade-offs of increasing process productivity by planning the design, or empowering the process capability for embracing change through the adoption of XP original design approach, without affecting product design quality.

Future work focuses on replicating the experimental studies presented in this paper, to get stronger evidence that can help us definitively discard quality differences between methods and get deeper insight about differences in productivity and quality.

References


