Quality, Productivity and Learning in Framework-Based Development: an Exploratory Case Study.

Maurizio Morisio
Politecnico di Torino, Italy
morisio@polito.it

Daniele Romano
Università di Cagliari, Italy
romano@dimeca.unica.it

Ioannis Stamelos
Aristotle University of Thessaloniki, Greece
stamelos@csd.auth.gr

Abstract

This paper presents an empirical study in an industrial context on the production of software using a framework. Frameworks are semi-complete applications, usually implemented as a hierarchy of classes. The framework is developed first; then several applications are derived from it. Frameworks are a reuse technique that supports the engineering of product lines. In the study we compare quality (in the sense of rework effort) and productivity in traditional and framework-based software production. We observe that the latter is characterized by better productivity and quality, as well as a massive increase in productivity over time, that we attribute to the effect of learning the framework. Although we cannot extrapolate the results outside the local environment, enough evidence has been accumulated to stimulate future research work.

Keywords: application framework, framework, product line, process quality, software reuse, empirical study, learning.

1 Introduction

Project managers and software engineers have often experienced a sense of frustration when a function, module or application must be developed that is similar to previously-developed ones. In fact, the concept of software reuse is nearly as old as software itself.

Earlier reuse models promoted the development of small-grained reusable components (for instance, procedures or classes) and their insertion in a reuse library. This model proved effective but limited because applications were able to reuse small-grained components only -- reuse occurs at the code level.

Application frameworks are a more promising reuse model, which aims to reuse larger-grain components and high-level designs. Based on object-oriented technology, they are defined as “semi-complete applications that can be specialized to produce custom applications” (Johnson and Foote 1988). Usually a framework is made up of a hierarchy of several related classes.

With the function or class library model, one or a few isolated classes (or functions) are reused. With frameworks, reuse takes a completely different form, with much higher leverage. The application is built by slightly modifying the framework, and by accepting in toto the framework’s high-level design. In other words, most of the framework is reused, and reuse takes place both at the design level and at the code level. As part of this approach, the high-level design of the framework establishes the flow of control that has to be accepted by the application. This is called “inversion of control”, since in traditional development the flow of control is defined by the application.

Framework-based development has two main processes: development of the framework and development of an application adapting the framework.

Developing a framework is more demanding than building an application. The designer should have a deep understanding of the domain the application is embedded into, and should anticipate the needs of future application developers. Domain analysis techniques, focused on modeling the scope of the domain, and commonalities and variabilities of applications, have been developed to guide the definition of framework requirements (Coplien et al. 1998). As a matter of fact, successful frameworks have evolved from re-engineering long-lived legacy applications, abstracting from them the knowledge of principal software designers.
Developing an application from a framework (provided this is feasible, i.e. most of the requirements of the application are satisfied by the framework) is a differential activity: the framework is parameterized, modified, extended.

The advantages promised by framework-based development -- better reliability, shorter time to market, lower cost -- depend on *not* writing software. However, the framework has to be developed first, and this means that several conditions have to be met:

- The commitment by management to commence and sustain investment in framework development.
- The stability of the domain on a time range long enough to have a positive return on investment.
- The availability of highly-skilled software designers and domain experts to build the framework.

Once the framework is available, developing an application from it means saving considerable time and effort compared to the development of the application from zero. However:

- Software engineers have to be trained to use the framework.
- Highly skilled designers have to maintain the framework.

Currently there is very little quantitative evidence to support project managers in decisions about framework-based development. What are, in quantitative terms, the gains in productivity, quality and time to market when using a framework? How many applications have to be developed before the investment in developing the framework is paid off and break-even point is reached? How quickly can a programmer master a framework he/she has not developed?

The contribution of this paper is in the quantitative analysis of productivity and quality in framework-based development, in comparison with traditional development. Specifically, we planned a study where a single programmer, a novice at the framework, has produced five applications fully based on a framework and four applications based on traditional component development. For each application we measure effort to develop it, effort to correct defects, size, complexity and reuse level; we define two indices for productivity and quality of the development process and observe their mean figures and trend over time, providing also empirical predictive models. We have found that productivity and quality are substantially higher in framework-based development than in traditional development. We also report gains in productivity and quality over time in both development modes. We attribute this effect to the fact that the programmer learns more and more about the application domain and the tools he uses. In particular, productivity improves in an impressive way in framework-based development.

Ideally, such a case study should have been performed involving several randomly selected programmers to account for variability within the programmers population. This study, however, was made in an industrial environment: the resources needed for observing a sample of programmers over a time long enough to assess programmer’s learning were not available. The results of our paper are based on the performance of a single programmer. Of course, this is of no guarantee that the results of our study generalize to larger groups. On the other hand a merit of exploratory studies like this one is to stimulate novel research hypotheses which may be then validated by more thorough and extensive investigations.

The paper is organized as follows: Section 2 describes the context of the study (the framework and the development processes used). Section 3 describes the empirical study, including goals and hypotheses, nature and criteria of the experimental design selected, validity threats, analysis and interpretation of results. A discussion in Section 4 concludes the paper.

### 1.1 Related work

We are aware of very few empirical studies regarding framework-based development.

(Matsson 1999) compares the effort to develop the framework and the effort to develop an application from it. Thirty one applications were developed from four subsequent versions of the framework, the application development effort using the framework was lower than 2.5% of the effort to develop the framework itself in more than 75% of the cases. He concludes that framework technology provides reduced effort for application development.

(Moser 1996) proposes the System Meter approach to estimate the effort involved in software projects, and compares it with Function Points on a set of 36 projects. A subset of these projects developed and used frameworks. The authors analyze the productivity for this subset and observe that productivity does not change remarkably if the size of the framework is omitted and only newly-developed code is considered. This is the same philosophy of measuring size and productivity that we used in our study. However, our result is different, as net productivity (after learning) is higher for framework-based development than for traditional development.

(Shull 2000) studies the process of using and learning a framework to develop graphical user interfaces by students in an academic setting. He proposes that techniques based on examples are the most suitable for supporting learning, especially for novice users.
In other empirical studies not related to frameworks (Prechelt 1998, Porter 1995, Porter 1997), the authors consider learning as a perturbing factor and design the experiment to neutralize its effect.

Although the general problem of deciding whether to develop a framework or not should be addressed with the aid of economic models, we are not aware of models developed specifically for frameworks. However, existing economic models for reuse (Poulin 1997b, Favaro 1998) could be adapted.

(Srinivasan 1999) points out that frameworks require a steep learning curve on the part of the reuser, something which has to be considered as another form of investment.

2 The framework and the application-development process

The network division of a research and development company has identified many domains and sub-domains where the commonality between applications is high, and therefore reuse is a potential choice. Among the several reuse techniques which have been tried, frameworks have proved to be a very promising and feasible option.

The division decided to set up a study on framework-based development to obtain quantitative insight on the process. Specifically, the goal was to build models (cost, return on investment, reliability, learning curve) in order to guide technical and business choices in the years to come.

In the following we describe the framework, the application-development process based on it, and the applications developed during the study.

2.1 The framework

The framework supports the development of multimedia services on a digital network. It uses a CORBA infrastructure, is developed in Java, and integrates some COTS (Commercial Off-The-Shelf) products.

The framework (see Figure 1) is composed of two layers. The lower layer, composed of service platform and special resources, offers network functionalities through CORBA APIs. The higher layer, organized in components, abstracts control services.

The service platform offers stable, basic functions for service access (request to activate a service session, request to join a session), management of profiles of service subscribers, session control (coordination of the resources used in a session), network resource control.

![Figure 1. Framework structure.](image)

Special resources offer specific functions (e.g. reflector [receives multimedia data packets from several sources and resends them to a set of destinations], video bridge [abstracts a hardware bridge for video data], vocal gateway [abstracts a hardware gateway for audio data]) -- each one with a dedicated API. This design allows the addition of special resources to the framework when needed, without changes to the service platform.

The components in the higher layer, implemented as several Java classes, belong to these families:
UAP (User Application): UAPs are executed on the terminals of the service’s participants; they implement the connection between terminal and network services. Given a service session, several UAPs can be activated, one for each role played in the service.

GSS (Global Service Session): GSSs contain the specific network logic (e.g. access and roles of participants, coordination of special resources) required by a service. Given a service, a single GSS is activated on the server of the service retailer.

SP (Service Profile): SPs describe each participant in a service session. At least one SP for each role is activated for a session; they all reside at the server of the service retailer and are persistent.

The framework is designed to be reused black-box (Fayad and Schmidt 1997), except for GSSs that have to be specialized from the base class (white-box). The framework is composed of 22 classes, or around 10KSLOC (source lines of code) and was developed before this study. A 30-page document describes the framework and is available to its users. The document contains a description of the components (UAP, GSS and SP), how they work together, and how they should be used to develop a service. In addition, an example of a service derived from the framework is provided. The formalisms used in the document are natural language, message sequence charts, hierarchy diagrams and composition diagrams.

2.2 The application-development process

As already mentioned, the application framework was developed before the present study, which deals exclusively with using the framework. Since framework-based application development is different from traditional application development specifically in the process used, the activities involved when employing the framework are listed below.

- **Requirements definition.** The requirements of an application are specified by means of a Use Case format. Tool used: MS Word.

- **Analysis.** Starting from the requirements and the Use Cases, the analyst defines how to implement the application using the framework. In the simplest case, this means selecting which component from the families UAP, GSS, SP should be reused and integrated into the framework. In most cases this will not be sufficient, so new components will have to be identified, specified, developed. Tool used: MS Word.

- **Component development.** This activity can occur only if the previous one has identified new components to be developed. As already mentioned, a component is a set of Java classes. Starting from text specifications developed in the activity Analysis, the component is designed; an IDL interface is specified; classes are coded and tested; the whole component is integrated and tested. In actual fact, this activity consists of a traditional development process. The process is independent of the framework: the developer receives the component specifications and develops the required Java classes by abstracting from the framework. Tool used: Java development environment.

- **Application development.** The framework is parameterized, after which the components identified in the activity Analysis and possibly developed in Component development are parameterized and integrated into the framework. The application is then tested informally. Tool used: Java development environment.

- **Acceptance test.** Test cases are generated from the use cases in the requirement document and subsequently these test cases are applied. Tool used: Java run time environment, Orbix-web.

- **Usage.** The application is used in the field; failures are logged in failure reports. Tool used: Java runtime environment, Orbix-web.

- **Corrective maintenance.** Failures reported from usage are repaired. Tool used: Java development environment.

3 The empirical study

The description of the present study follows as far as possible a general template suggested by Wohlin et al. (2000), which summarizes the manner in which most empirical studies in software engineering are presented: definition, planning, validity, and statistical analysis of results. Each topic contains in turn a number of issues. The first subsection, Definition, states the goal of the study (object and objectives), discusses its nature (case study vs. experiment) and describes the context (programmer and his environment). Planning comprises the formulation of the research hypotheses, the selection of independent and dependent variables with a justification of the metrics adopted to express them, the plan (sequence of developed applications plus their description) to be adopted and its design criteria. Validity
discusses construct, internal and external validity. In *Statistical analysis* the statistical analysis performed on the results (descriptive statistics, graphical representation, regression models) is described.

### 3.1 Definition

#### 3.1.1 Goal of the study

A general statement of the goal of our study is: To analyze the development process of framework-based applications with respect to development without the framework for the purpose of evaluating productivity, quality and learning effect in the context of web-based multimedia services developed in an industrial setting by programmers who were not the framework developers.

#### 3.1.2 Nature and context of the study

The study described here is very much a compromise between a controlled experiment and a case study. The user was well aware of possible limitations and validity threats (see section 3.3) because of his decision to perform the study inside his own company under a rather severe budget limit and within a short time frame. Since the purpose of the present research is to assess the performance of developers who have little or no prior knowledge of the framework, an improvement in productivity and quality over time is expected, and therefore investigating the learning effect is a primary objective of the study. This is not a purely academic problem, since a steep learning curve can be a decisive factor in making framework technology profitable, as already documented elsewhere (Srinivasan 1999). Given the time and cost constraints of our situation, we decided to use only one programmer and observe him during the development of a number of applications (nine). The underlying rationale is this: whenever significant changes over time occur in individuals while they are receiving different treatments, it can be more informative to study one subject for several hours than several subjects for one hour (Kratochwill 1992). Therefore we favored control over the development-mode factor and investigation into the learning effect, sacrificing generalizations regarding programmer population. Furthermore, this is a typical situation, since in companies programmers who develop and maintain applications based on a framework tend to do this many times. We chose a computer science graduate with a good knowledge of application domain and expertise in object-oriented programming, a beginner in the language used (Java), a novice to the framework. We will discuss this issue further in the forthcoming *Validity* section.

The developed applications belong to either a control group (development without framework, corresponding to the activity Component Development in the process described in section 2.2) or to an experimental group (development with framework, corresponding to all other activities in the process in section 2.2). They have all been developed through the same process, described in section 2.2. However, for those in the control group, the effort spent on the activity Component Development is predominant -- accounting for the large majority of the effort spent. Such applications can be considered as variants of an application previously developed. The requirements and design here are basically the same as a previous application, but the new functionality requires the development of various new classes, which can be accomplished through the activity Component Development. Component Development is basically a traditional development process, where code is developed and tested starting from certain specifications, with no reference to the framework. We verified this point by specifically interviewing the programmer during and after the development of the applications in order to ensure that component development occurred without any reference to or use of the framework. For these reasons, we have used applications where Component Development is the predominant activity as a control group. The characteristics of the two subgroups are summarized in Table 1.

### Table 1. Characteristics of applications in experimental and control groups.

<table>
<thead>
<tr>
<th>Experimental Group (Development with framework)</th>
<th>Control Group (Development without framework)</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦ Applications where Activity Component development is absent.</td>
<td>♦ Applications where Activity Component development is predominant.</td>
</tr>
<tr>
<td>♦ Other activities make explicit and continuous reference to the framework.</td>
<td>♦ Component development makes no reference to the framework.</td>
</tr>
<tr>
<td>♦ Effort spent in Component Development is zero.</td>
<td>♦ Effort spent in Component Development is non-zero.</td>
</tr>
<tr>
<td>♦ Only effort spent in other activities is considered.</td>
<td>♦ Effort spent in other activities is minimal and is excluded from the analysis.</td>
</tr>
<tr>
<td>♦ The application is developed independently of other</td>
<td>♦ The application is a variant of another previously developed.</td>
</tr>
</tbody>
</table>
applications. A variant may be developed from it later. This implies a precedence in development.

The present study can be defined as a single subject (or single case) experiment according to Harrison (2000), who refers to a vast practice of such experiments in social science and medicine and promotes their use in software engineering. The present study could also be denoted as a multi-object variation study—adopting the classification proposed by Basili et al. (1986) and Wohlin et al. (2000). These latter argue that our type of study is a quasi-experiment because the subject is fixed instead of being randomly selected across the projects, as should be the case in a canonical experimental set-up. However, the software engineering community is somewhat reluctant to call an “experiment” an empirical study that (even if it includes control factors), involves only one developer, since there is generally considered to be a wide variation in performance among programmers. Therefore we call our study an exploratory case study with a single subject.

3.2 Planning

3.2.1 Research hypotheses

The research goal is deployed in the following four hypotheses to be tested.

**H1 Development with framework provides higher net productivity than development without framework.**

This statement should not be read in a trivial sense: i.e. reusing a framework shortens the time needed to produce an application. Of course it does, because a smaller amount of code needs to be written thanks to reuse. This is the basic assumption upon which the convenience of framework-based development lies (however, in certain contexts, increased reliability might justify the use of a framework even with a lower gross productivity). The hypothesis to be verified is whether, given the same amount of time, the programmer is able to write a larger piece of code if he uses the framework. In this sense the hypothesis is not trivial, as it assumes that a framework enables the programmer not only to write less code but also to do so more quickly. Therefore net size is considered, namely the amount of additional code actually written in order to customize the framework and turn it into a specific application.

**H2 Learning increases net productivity in development with framework more than in development without framework.**

We expect an increase in productivity over time, no matter which development mode is considered. Here we are testing whether the productivity gain is faster in framework-based development than in traditional development. This would imply a competitive advantage in the long run, after the initial expenditure for framework development is paid back.

**H3 Development with framework is less prone to failures than traditional development.**

In the same fashion as for H1, this is meant in a non-trivial sense. It is quite obvious that framework-based development, as for any reuse technique, means writing less, and reusing more documents and code, than with traditional development. Since framework modules are reused and field-tested in several applications, we expect that—overall—the failure density of the application is lower. However, the hypothesis means that if we compare two pieces of code of the same size developed (not reused) in the two modes, we find fewer errors in the framework mode. The rationale here is very close to that of H1, for it is reasonable to assume that an easier-to-write code will contain fewer errors.

**H4 Learning reduces failure occurrence in development with framework more than in development without framework.**

In general we expect that failure density should decrease over time thanks to a learning effect; the hypothesis assumes that, in framework-based development, failure density decreases over time more quickly than in development from scratch. The rationale is the same as for H2.

3.2.2 Selected variables and metrics

The developed applications are characterized by a set of attributes, which are the independent variables of the study. They are: development mode, application domain, programming language, application size, cumulated size (from the first up to the i-th application), complexity, reuse level of the framework. Some of these variables are controlled; some are not. The controlled variables are development mode, application domain and programming language. The development mode is an experimental factor with two levels (with/without framework). Application domain (multimedia...
networking services) and programming language (Java) are fixed in the present study. Admittedly, fixed variables confine the scope of the investigation, but they do not introduce possible colinearities with any other independent variables which would be responsible for confounding of effects. Uncontrolled variables (called also covariates) are: individual size, cumulated size, complexity and reuse level of the framework. Being measured a posteriori, their effect on performance can also be analyzed. In the next paragraph the factor, the covariates and their metrics are defined. Table 2 summarizes the variables, adding mathematical definitions and units.

- Development mode, \( F \). It is a two-level qualitative factor. Levels are development without the framework, coded with 0, and development with the framework, coded with 1.
- Net size of an application, \( S \). It is a covariate because its value is unknown before development. It considers only newly-written code, thus excluding reused code. It is expressed as Object Oriented Function Points, (OOFP, Antoniol et al. 1999), which compute the functionality starting from specific classes and methods. As plausible metrics, initially we also considered Function Points (FP, Albrecht 1979, IFPUG 1994), System Meter (Moser 1996) and the classical Source Lines Of Code (SLOC). We excluded FPs because counting rules were not appropriate for Web-based object-oriented applications and because they do not allow the distinction between functionality provided by the framework and that provided by the code developed for the specific application. This leads to the conclusion that Function Points cannot be used to measure the net size of an application developed, nor, as a consequence, the reuse level (Poulin 1997). We excluded SLOCs because they have been heavily criticized for causing practical difficulties when used to measure productivity (Jones 1991). We preferred OOFPs to the System Metric mentioned above because OOFPs are more recent, follow the concepts of the well-known FP and are more straightforward to apply. The reader may refer to (Morisio et al. 1999) for more details about measurement activities and for a discussion regarding measurement of the reuse level.
- Cumulated net size, \( L \). It is a covariate that sums up the net size of applications developed up to the current one. It is introduced to express the effect on the performance of the programmer’s learning over time. See Construct Validity later on for a discussion of its ability to produce the desired effect.
- Complexity, \( C \). It is a covariate measured after development as the number of methods per unit net size. It was inspired by the WMC measure in Chidamber and Kemerer’s (1994) metric suite. WMC considers each method as a complexity unit at the level of a class. In our study, we consider each method as a complexity unit at the level of the entire application.
- Reuse level, \( RL \). We use the traditional definition for reuse level (Poulin 1997), or the ratio between the size of what is reused from the framework and the total delivered size in an individual application. Ideally the framework is reused in toto by each application. In actual fact, each application reuses part of the framework but not all of it. The difficulty is that measuring the exact amount of framework reused by each application is impractical. We introduce the utilization factor \( U_{FWK} \), with range \([0,1]\), \( U_{FWK} = 1 \) means the framework is reused in toto) into the definition of the reuse level. Normally it should be a covariate, measured ex-post for development with the framework (being zero for development without the framework). However, in our case reuse level varies minimally within framework-based applications (from 77% to 85%, assuming \( U_{FWK} = 1 \) and therefore it is practically an alias of the development mode factor. For this reason it is excluded from the analysis.

Two dependent variables, responses, are of interest in the present study (cfr. Table 2 for more details).
- Productivity, \( p \). It is calculated as the net size of an application divided by the effort required for its completion. Please notice that we use net size at the numerator, so productivity should actually be called net productivity -- we omit the adjective merely for the sake of simplicity. The development effort is measured by the programmer, who logs it on a daily basis.
- An index of quality of programming, \( q \). It is defined as the relative deviation between development effort and the rework effort required to correct code failures identified by the acceptance test (its range is 0 to 1, 1 means no failure encountered; 0 means that rework effort for correction equals development effort).

For ease of interpretation, both responses are defined as the larger the better.
Table 2. Factors, covariates and responses in the experiment.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Definition</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>Factor</td>
<td>Indicator variable for development mode (0 means without framework, 1 with framework)</td>
<td>Dimensionless</td>
</tr>
<tr>
<td>$S$</td>
<td>Covariate</td>
<td>Net size of developed applications</td>
<td>Object Oriented Function Points (OOFP)</td>
</tr>
<tr>
<td>$L$</td>
<td>Covariate</td>
<td>Cumulated size of developed software up to the i-th application (proxy of programmer’s learning)</td>
<td>Object Oriented Function Points (kOOFP)</td>
</tr>
<tr>
<td>$C$</td>
<td>Covariate</td>
<td>Number of methods ($n$) per unit size in the developed application (proxy of application’s complexity)</td>
<td>OOFP$^{-1}$</td>
</tr>
<tr>
<td>$RL$</td>
<td>Covariate</td>
<td>Reuse level, ratio between reused size and total size delivered. Applies only to applications developed using the framework. Each application reuses a certain part (utilization factor $U_{FWK}$) of the framework ($S_{FWK}$) and develops new parts ($S_i$)</td>
<td>Dimensionless</td>
</tr>
<tr>
<td>$p$</td>
<td>Response</td>
<td>Productivity of programming calculated as ratio between net size $S$ and time spent developing (effort, $e$) the application</td>
<td>OOFP $\cdot$ h$^{-1}$</td>
</tr>
<tr>
<td>$q$</td>
<td>Response</td>
<td>Index of quality of programming calculated as relative deviation between development effort and the rework effort required to correct failures detected during the acceptance test ($e_{rew}$)</td>
<td>Dimensionless</td>
</tr>
</tbody>
</table>

$$L_i = \sum_{k=1}^{i} S_k$$

$$C_i = \frac{n_i}{S_i}$$

$$RL_i = \frac{U_{FWK} S_{FWK}}{(U_{FWK} S_{FWK} + S_i)}$$

$$p_i = \frac{S_i}{e_i}$$

$$q_i = \frac{(e_i - e_{rew,i})}{e_i}$$

3.2.3 Experimental plan and design criteria

The overall experiment plan is depicted in Table 3. Applications in the control group appear in the far right column, the others in the middle column. Run order increases from the top down. The time schedule and the effort figures of the developed applications are reported in Figure 2. The whole development process was completed in a six-month period with a one-month vacation interruption. The programmer worked nearly full time on this project.
Table 3. Experimental plan.

<table>
<thead>
<tr>
<th>Run order</th>
<th>Application (Experimental group)</th>
<th>Application (Control group)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Spy Camera (SPY)</td>
<td>Spy Multi</td>
</tr>
<tr>
<td>2</td>
<td>Telelearning (TLL)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MultiConference (MCS)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Auction (AUC)</td>
<td>AUC Browser</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Video on Demand (VOD)</td>
<td></td>
</tr>
</tbody>
</table>

A brief description of each application, in the same order as developed, is given below for interested readers.

1-SPY – Spy Camera – This application permits the monitoring of a number of video cameras on different sites. The observer starts the application, specifying the sites where the cameras are located and then connecting/disconnecting with them. The site must accept connection or disconnection by the observer.

2 TLL – Telelearning – This application permits the setting up of a distance learning session. The roles involved are teacher and students. The teacher starts and ends a lesson. A student can join / leave a lesson upon permission from the teacher. The teacher can decide to test a student (in this case all other students are simply observers) and to exclude a student from the session. In addition, a student may privately pose questions to the teacher.

3-SPY Multi – Spy camera multi-observer. Variant of Spy camera. Same as Spy camera, but there may be several observers during a single session.

4-MCS – Multiconference – Audio and video multiconference on the Internet. The roles are chairman and participant. All roles can see and hear other participants. The chairman starts and ends the session, calls participants to join, accepts or refuses requests to leave from participants. The chairman moderates the session by giving the floor to one participant at a time. He is allowed to pass his role to another participant.

5-AUC – Auction – This application offers an Internet-based auction. The roles are auctioneer and bidder. The auctioneer starts and ends the session. He auctions one item at a time and regulates bids, assigning the item to the best bid, or possibly retiring it from the auction. A bidder can exit the session only if his bid is not the highest one at a certain moment.

6-AUC Browser – Auction with browser – Variant of Auction. The only difference is that when the first item is auctioned, an internet browser is started on each bidder’s terminal, open at the URL where the item is described.

7-MCS Mail – Variant of Multiconference. In MCS, if the chairman calls a participant and cannot reach him, nothing further occurs. In this variant, an e-mail is sent to the participant to notify him of the unsuccessful attempt to reach him.

8-MCS Sms – Variant of Multiconference. An SMS message is sent when the participant cannot be reached by the chairman.

9-VOD – Video on demand. The roles are customer and provider. The customer requests commencement of a session, selects a movie from a catalog, pays. The provider checks these operations; when completed successfully, he starts the movie for the customer. The provider ends the session when the movie ends; the customer can pause/resume the session, or stop it.

The time sequence of the applications is a compromise between the randomization principle commonly used in Design of Experiments and the practical needs of the study. A randomized run order is normally adopted, as it ensures that time order neither masks existing factor effects nor it erroneously reveals not existing ones (Box et al. 1978). In our case this precaution is even more vital, since we are interested in investigating the learning effect. For example, had we adopted the sequence 111110000 (five applications with framework, followed by four without), the effect of factor F (development mode) would have been confused with the learning effect because the last four developments (F=0) would have benefited from a larger learning than the five initial ones (F=1). However, a complete randomization could not be applied in our study for two reasons: first, we had the obvious constraint that variants had to be developed after their parental applications; second, we deliberately planned to develop the two most similar framework-based applications (SPY and VOD) at the beginning and at the end of the experiment. In this way the estimate of the learning effect in framework-based development (one of our major concerns) should have increased precision because it exploits the largest possible leverage of learning (likewise a larger range of the x independent variable increases precision of the estimated slope of y(x) when fitting a straight line to data using simple linear regression).

We did not include replicated runs in the experiment, although they are quite valuable to provide an external estimate of the random scatter of responses. We believe that replicating the development of an application may be
counterproductive in the case of a single programmer. In fact, the second time a programmer develops an application, he is faster not only because of learning, but also because he remembers the design and code already developed and this would positively bias his performance.

Figure 2. - Time schedule of the developed applications, with framework (white boxes) and without framework (gray boxes). Effort figures, in hours, are marked inside the boxes. Month 3 was a vacation period. Note that for applications without framework the figures refer only to component development.

3.3 Validity

We discuss here the validity of the experiment. In the same manner as (Judd 1991, Wohlin 2000), we consider construct validity, internal and external validity.

3.3.1 Construct validity

Construct validity aims at assuring that the metrics used in the study reflect real-world entities and attributes reliably. The key attributes here are productivity, quality and learning. Measures selected for size (OOP) and complexity (number of methods per unit size) have already been justified.

For productivity, we use net productivity, i.e. the ratio of the size of software newly-written for an application, and the effort to develop the application. Another option is to use gross size (size of the framework + net size) at the numerator, the rationale being that an application delivers the whole functionality in the framework, and not only what is newly-written. Symmetrically, the denominator should consider the effort required to develop the framework. Since, however, the framework is reused in several different applications -- with different reuse levels -- it is not clear how to distribute among them the effort expended in developing the framework. What is more, the framework and applications are developed by different programmers, so it would not be correct to mix their productivity levels. For these reasons we use net productivity. Naturally, the investment needed to develop the framework should also be taken into account in a comparative economic evaluation.

For quality, we use an index of rework effort. Although using the number of defects could have been an alternative, we chose rework effort because it can be neatly integrated into the overall effort figure, thus providing a consistent and complete base on which to build a cost model for break-even point prediction.

The metric used to capture the learning effect is a delicate issue. We measured the learning effect on performance (productivity or quality) at a given time as the improvement of that performance (with respect to the beginning of the development) -- due solely to the fact that our programmer had already developed a certain size of code previously. To understand our approach, it is useful to introduce here the full model used to describe the experimental responses:

\[ y = \alpha_0 + \alpha_1 F + \alpha_2 L + \alpha_{12} FL + \alpha_3 S + \alpha_4 C + \varepsilon \]  

where \( y \) is either productivity or quality index, coefficients \( \alpha \) are the parameters to be statistically estimated from data and \( \varepsilon \) is the experimental error, assumed to be normally distributed with zero mean. The learning effect on \( y \) up to the total developed size \( L \) is:

\[ \alpha_2 L + \alpha_{12} FL = (\alpha_2 + \alpha_{12} F)L \]
which is $\alpha_2 L$ for non-framework-based development and $(\alpha_2 + \alpha_{12} L)$ for framework-based development. In reality there are three components of learning, namely learning the programming language, the application domain, the framework usage. Learning the programming language and the application domain occur in both development modes, while improvement in use of the framework is obviously specific to framework-based development. Therefore we can legitimately associate the common term $\alpha_2 L$ with non-specific learning and the differential term $\alpha_{12} L$ with framework-specific learning. That is why it is vital to include the interaction between $L$ and $F$ in the model.

A good measure of learning is one that documents the actual rate at which the programmer understands concepts during his work; unfortunately, however, a measurement tool of this nature is extremely difficult to find. One option is periodical tests or possibly interviews with the programmer; these are rather complicated to manage and subject to errors. Monitoring the subjective feeling of the programmer might be an option; this is less complicated to manage, but even more error-prone. If we take the example of manufacturing, the learning of a worker in doing a repetitive job is measured by the reduction in the time required to do the same task. We used an approach similar to the manufacturing situation and chose as a proxy of learning the cumulative size of software written by the programmer. This has the advantage of being objective and easy to collect. While we realize that this is not the optimal choice, since in our case the task is not completely repetitive, it is definitely cost-effective.

3.3.2 Internal validity

Internal validity discusses whether the experimental design allows the demonstration of causality between input variables (factors and covariates) and responses.

The design isolates one factor and considers three covariates. The primary concern focuses on the effects of the factor, the learning covariate and their interaction; the other two covariates (size and complexity) are added merely to improve precision in the estimation of the three above effects. We wish to point out that hypotheses H1 and H3 are related to testing whether the effect of $F$ is significant and positive (i.e. $\alpha_1 > 0$ in the models for productivity and quality); hypotheses H2 and H4 are related to testing whether the effect of $FL$ is significant and positive ($\alpha_{12} > 0$ in the models).

The number of runs (nine: four in the control group, five in the experimental) was chosen in order to accommodate models with five terms at most, excluding the constant, (see model (1)) -- equivalent to about one half of the total degrees of freedom. However, even more parsimonious models may be expected because the size variable, $S$, might be not statistically significant, having already been accounted for in the definition of the responses (explicitly in productivity and implicitly in the quality index). This is likely to reduce the number of active predictors in the fitted models.

Design criteria have already been discussed.

3.3.3 External validity

External validity discusses how well the study results can be generalized.

Our study is an exploratory case study on a single subject. We needed to observe the same programmer several times in order to make inferences about learning. Having only one programmer involved did not present us with an ideal situation. On the other hand, replicating the study with more subjects was considered too expensive by the company.

- On the negative side, we have no quantitative, objective means to evaluate our programmer in comparison to a population of programmers even though it is well known that individual personality characteristics can have huge effects on the products and process of programming. In the subjective judgment of the authors, our programmer is probably above-average but is not unrepresentative of the general programmer population.
- On the positive side, the study is comparative: the same programmer is used to compare two ways of programming. In principle, there is no reason why he should favor either one of them.

Overall, we cannot claim that our results remain valid for all programmers. However, we believe to have provided an empirical base for discussion which has been missing so far, and hope to stimulate interested researchers to make confirmatory experiments or other related studies in the future, since advances in empirical research are nearly always incremental.

The study is limited to a specific programming language, application domain and framework. The programming language employed has become increasingly popular, and one can argue that it is representative of the class of object-oriented programming languages widely used in practice. The application domain is also increasingly popular, but clearly has a number of specific characteristics that could influence results.

The size and design of the framework could also influence results. The design is as simple as possible, and probably representative of a large family of frameworks. The framework is made up of 22 classes, or slightly less than 10Kloc.
We believe it can be classified as a small-to-medium framework. We believe that large frameworks (100 classes and more) have different effects on learning: learning duration may well be longer, and the variation in productivity for a programmer after learning could be higher (more leverage from a larger framework) or lower (more difficulty in managing a larger framework) than in our case. It should be noted that, given a complex domain, the trend is to develop several related but smaller frameworks rather than developing a single large framework.

Finally, reuse level does not vary considerably in the present study -- which explains its exclusion from the model. We feel comfortable in stating that this does not influence results when the reuse level is very high, as in our case. When the reuse level is lower (especially below 50%), results may well be different. However, reusing a framework is not usually justifiable when the reuse level is so low. An issue related to reuse level is selection of the applications. Ours were selected so as to be suitable for implementation with the framework (i.e. so that they had a high reuse level). We believe this is not a threat to the analysis and extrapolation of results, since it reflects a common practice with frameworks: first a framework is developed when it is likely that several applications can be derived from it; then the framework is used whenever it is suitable for supporting an application.

3.4 Statistical analysis

Hereafter we present the empirical results and the statistical analysis performed on them. The Minitab package (Minitab 2000) has been used for the analysis. Data referring to independent variables and responses resulting from the study are reported in Table 4. Simple graphical displays of the two responses using dot-plots (Figure 3) and dispersion plots versus cumulated size (Figure 4) reveal the fundamental features of results. First we make a qualitative evaluation of the four hypotheses by looking at the diagrams. Later, we perform statistical tests to give formal assurance. Model (1) is also estimated for both productivity and quality.

Table 4. - Measures of inputs and outputs of the study. Data refer to new documents and code, and do not consider documents and code in the framework.

<table>
<thead>
<tr>
<th>Application</th>
<th>Type</th>
<th>Classes</th>
<th>Methods</th>
<th>Size</th>
<th>SizeReuse (with $U_{FWK} = 1$)</th>
<th>Learning Complexity</th>
<th>Effort</th>
<th>Rework effort</th>
<th>Productivity</th>
<th>Quality index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>$F$</td>
<td>$n$</td>
<td>$S$</td>
<td>$RL$</td>
<td>$L$</td>
<td>$C$</td>
<td>$e$</td>
<td>$\varepsilon_{rew}$</td>
<td>$p$</td>
<td>$q$</td>
</tr>
<tr>
<td>SPY</td>
<td>1</td>
<td>13</td>
<td>39</td>
<td>1718</td>
<td>247</td>
<td>0.82</td>
<td>0.247</td>
<td>0.16</td>
<td>95</td>
<td>11.0</td>
</tr>
<tr>
<td>TLL</td>
<td>1</td>
<td>13</td>
<td>60</td>
<td>2673</td>
<td>331</td>
<td>0.78</td>
<td>0.578</td>
<td>0.18</td>
<td>87</td>
<td>13.0</td>
</tr>
<tr>
<td>SPY Multi</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>603</td>
<td>19</td>
<td>-</td>
<td>0.597</td>
<td>0.16</td>
<td>18</td>
<td>6.5</td>
</tr>
<tr>
<td>MCS</td>
<td>1</td>
<td>8</td>
<td>37</td>
<td>1826</td>
<td>204</td>
<td>0.85</td>
<td>0.801</td>
<td>0.18</td>
<td>45</td>
<td>5.5</td>
</tr>
<tr>
<td>AUC</td>
<td>1</td>
<td>15</td>
<td>59</td>
<td>3020</td>
<td>341</td>
<td>0.77</td>
<td>1.142</td>
<td>0.17</td>
<td>63</td>
<td>6.5</td>
</tr>
<tr>
<td>AUC Brow</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>362</td>
<td>17</td>
<td>-</td>
<td>1.159</td>
<td>0.18</td>
<td>13</td>
<td>3.0</td>
</tr>
<tr>
<td>MCS email</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>256</td>
<td>26</td>
<td>-</td>
<td>1.185</td>
<td>0.15</td>
<td>16</td>
<td>4.0</td>
</tr>
<tr>
<td>MCS SMS</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>403</td>
<td>41</td>
<td>-</td>
<td>1.226</td>
<td>0.12</td>
<td>16</td>
<td>4.5</td>
</tr>
<tr>
<td>VOD</td>
<td>1</td>
<td>13</td>
<td>39</td>
<td>1618</td>
<td>247</td>
<td>0.82</td>
<td>1.473</td>
<td>0.16</td>
<td>34</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Figure 3. Dotplots of responses. Empty/filled circles indicate development without/with framework; numbers attached to circles refer to the execution order of experimental runs.

Hypotheses H1 and H3 are evaluated by comparing, in the two development modes, the average figures of productivity (H1) and quality (H3). In both cases (see Figure 3 and Figure 4) data points in the experimental subgroup are higher than those in the control subgroup. This gives us a clear graphical indication that both H1 and H3 are true.

Figure 4. - Dispersion plots of productivity (left) and quality index (right) as a function of the cumulated size of developed applications. Empty/filled circles indicate development without/with framework.

Regression lines for the two subgroups are also displayed.

Hypothesis H2 and H4 are evaluated by comparing, in the two development modes, the variation of productivity (H2) and quality (H4) over time (Figure 4 reports cumulated size on the x axis, but size cumulates with time). In Figure 4 regression lines fitting the data points are also plotted for ease of interpretation. Productivity, both with and without the framework, increases. But productivity using the framework increases more than productivity without it. This suggests that H2 is true. As far as the quality index is concerned, we can observe that there is an improvement over time in both cases, but improvement using the framework is now inferior to improvement without it. This suggests that H4 is false.

The analysis summarized in Figure 4 is simple and intuitive, but limited since it considers only cumulated size as an independent variable. A more rigorous evaluation of the hypotheses needs to consider all the input variables available in the study, as in model (1). Now we proceed with estimating the model, i.e. its parameters \( \alpha \).

Model estimation is carried out in two steps. First we identify, for both responses, the statistically significant predictors in model (1), then we fit the model estimating only the coefficients related to the significant predictors. Identification of significant predictors is performed by Analysis of Covariance (Mason et al. 1989), a statistical procedure which permits the simultaneous testing of the effects of factor \( F \), covariates \( S \), \( L \), \( C \) and interaction \( FL \) on responses. Notice that in our case we could not apply the more usual Analysis of Variance. In fact, the presence of covariates makes the design non-orthogonal and, as a consequence, contributions of all terms in model (1) to total variation of responses are not separate from each other. Therefore analysis cannot be conducted in a “one-shot” fashion as in the Analysis of Variance. Instead, Analysis of Covariance is basically a trial-and-error iterative procedure aiming at selecting the best subset of explanatory input variables; it can also be regarded as a procedure for identifying the best
form of a linear predictive model for an experimental response (Box and Draper, 1986). A summary of the Analysis of Covariance procedure is reported in the Appendix.

The outcome of the procedure is that for productivity, the 90% statistically significant predictors (in order of significance) are: FL, F, C, L. For the quality index, only F and L are significant above 90%. Finally, we build the models using least square linear regression on our data set. Note that parsimonious models will be obtained (four predictors for productivity, two for quality); this is an assurance on the goodness of the models as we are relying upon nine experimental runs only. The model for productivity is:

\[
\hat{p} = 3.28 + 1.54F + 1.11L - 18.4C + 2.46FL
\]  
(2)

with a standard error of estimate of 0.267 and an adjusted determination coefficient of 98.4%. Programming with the framework results in an average initial benefit in net productivity of 1.54 OOFP per hour, which grows very quickly thanks to a framework-specific learning rate which is more than double (2.46) that of the non-specific one. Comparison between net productivity in the two development modes, as predicted by the model, is depicted in Figure 5 (complexity is fixed at the sample mean, 0.17 methods per OOFP).

The regression model for the quality index is:

\[
\hat{q} = 0.627 + 0.191F + 0.0881L
\]  
(3)

with a standard error of estimate of 0.0338 and an adjusted determination coefficient of 89.0%. Programming with the framework results in an improvement of the q index of nearly 0.2. Notice that the interaction FL is not included in the model because its effect is not statistically significant over the 90% threshold selected. In fact, FL’s effect on quality is significant about 80% (in a model including F, L and FL). This implies that the q index rises at the same rate in the two development modes when L increases (Figure 5), differently from what Figure 4 suggested.

We can now analyze again the hypothesis, using models (2) and (3) and Figure 5. Values of both productivity and quality, for a fixed value of cumulated size, are always higher when using the framework (F=1). We can therefore conclude that H1 and H3 are true. To test H2 and H4 we have to look at the slopes of the lines. In the case of productivity, the slope when using the framework (F=1) is higher, so we can conclude that H2 is true. In the case of quality, the slopes with or without framework are the same, so H4 is false.

In practice, the discussion of the hypotheses has produced the same results, both using the plots of the data points (Figure 4) and the statistical models (Figure 5). Needless to say, statistical models have a higher degree of reliability than a simple glance at the data, both because models consider all the explanatory variables (only L is in Figure 4), and because they are estimated after testing the statistical significance of the explanatory variables. Incidentally, this is the reason why in Figure 4 the slopes relating to quality are different, while in Figure 5 they are the same: the model for quality does not include the FL term because not statistically significant.

Recently a critique of software defect prediction models has been published (Fenton 1999). This work concludes that it is difficult to predict defect density using size and complexity metrics alone. For example, the authors argue that the time dedicated to the test of each module is an important factor that should always be accounted for in building the model. In our case, however, this factor was considered during the design of the experiment and it was assured that the programmer tested both applications on an equal basis.
Figure 5 – Net productivity (solid lines) and quality index (dashed) vs. net code size in development with and without framework. Complexity is fixed at the sample mean, 0.17 methods per OOFP. After 1.47 kOOFP curves extrapolate experimental results.

4 Conclusions

The objective of our work was to investigate quality and productivity issues and the effect of learning in framework-based object-oriented development. Our motivation stemmed from the need to measure and understand the potential benefits that frameworks are supposed to provide. The lack of studies publishing data and evaluation results from framework-based projects was an additional stimulus.

In order to achieve our goal, we carefully designed an exploratory study using a single programmer in an industrial environment. The decision to use a single subject was dictated by constraints imposed by our industrial partner. The programmer produced five applications fully based on a framework and four applications based on traditional component development (control group). We defined metrics for productivity, quality and learning -- paying particular attention to consider net size only (i.e. newly developed code only, and not reused code). Based on measurements taken on the developed applications, a statistical analysis of the data collected was performed.

We observed that productivity and quality were substantially higher in applications developed using the framework than in applications implemented through traditional development. The general expectation when using frameworks is that there will be an increase in gross productivity (i.e. productivity including both the size of new code and the size of the framework). This expectation is quite intuitive; our study confirmed its validity. Further, the study demonstrated that also net productivity (i.e. productivity considering only new code developed around the framework) is higher. A less intuitive expectation is that implementing one unit of functionality (one OOFP in our case) around a framework is faster than implementing one unit of functionality without a framework. Again, the study confirmed the expectation’s validity. This finding can be explained by considering that a framework encodes the most difficult design and coding issues of a domain or sub-domain. The programmer reuses the difficult parts, and writes only the remaining easier parts. Framework users perform tasks that are more similar to parameter setting: compare, for instance, the task of designing and coding an algorithm to that of designing and coding of the data input/output functions of the same program. The latter task presents far fewer problems and “tricky points” than the former. On the other hand, the developer of a framework has to deal with the major design and coding problems and challenges. Therefore our hypothesis is that the productivity of framework developers is below average and substantially less than that of the framework users. A similar reasoning applies to quality (in our case, effort needed to repair defects). Quality is higher when reusing a framework because the more difficult tasks have already been performed by framework developers.

We also observed significant gains in productivity and quality due to a learning effect, or the improved skill of the programmer in performing a task, due to the repetition of the task over time. In particular, productivity exhibited an impressive learning effect for framework-based applications. This finding confirms the experience of anyone who has used a framework. The leverage they offer is high, but the quantity of knowledge to be digested before becoming a proficient user is huge. In other words, learning is the key and sufficient time should be allocated for it. A possible interpretation for this lengthy learning time is that in framework-based development a major conceptual form of learning
is involved. Borrowing from the organizational learning theory of (March 1996, Nonaka 1994), we distinguish two types of learning: operational and conceptual. The former deals with speeding up repetitive operations, the latter with acquiring conceptual knowledge. As frameworks encapsulate high-level knowledge, the conceptual component of learning is more prominent than in traditional development.

The issue of when learning has finished, and the best way of measuring it (in number of applications developed? in cumulative size developed?) remains open and requires new empirical studies.

Our data showed as well that learning had no effect on failure-occurrence rate in development with framework as compared to development without framework. Seemingly this result contradicts the previous one. However, a reasonable explanation is that the quality index is limited by 1 and, approaching this limit, room for further improvement becomes lesser and lesser. As a matter of fact, quality figures for framework development are considerably closer to unity than those of traditional development and this might limit the leverage of learning. Anyway, further studies are needed to clarify the issue.

There are certain limitations that allow the above conclusions to be generalized only to some extent. The major limitation is the employment of a single subject in the study. We judged our programmer as belonging to a slightly above average level of competence -- which does not allow, for example, the generalization of our findings in cases where particularly skilled subjects are employed in framework-based development. Other characteristics of our study that may limit conclusions are the specific framework size (small to average) and the specific application domain. However, we argue that framework size may not be so prohibitive as to prevent our conclusions from being of practical use, since in complex domains a reasonable strategy would be to employ more than one small framework instead of a single large one.

Though our findings cannot be said to be definitive, they do provide a basis for discussion and indicate future research directions. As an example, the replication of our study with more than one subject would produce a sounder basis for empirically derived conclusions about framework-based development. We anticipate, however, that such an experiment will demand significantly more resources than our study had available. Experiments investigating important issues such as framework size, programming language, application domains, etc. are also needed. Finally, empirical investigation of the framework investment break-even point is necessary, in order to provide managers with important decision-making tools which may assist the spread of such a promising technology. We are currently working on this issue.

5 Acknowledgements

The authors would like to thank Fabio Balduzzi, Corrado Moiso and Bruno Noero for initiating the study, Shari Lawrence Pfleeger, Forrest Shull, the anonymous referees and Don Batory for reviewing the paper and contributing to improve it.

6 References


7 Appendix

In Tables A1 and A2 you will find a summary of the Analyses of Covariance for productivity and quality, including all terms (covariates S, L, C, factor F and interaction FL). In productivity one effect stands out – FL –; one is not statistically significant – S –, while the others are questionable (Table A1). Pooling into error the non-significant effect, the new Analysis of Covariance in Table A3 shows that the significance of all remaining effects is increased; a confidence limit superior to 95% holds for FL, F and C (in order of importance), superior to 90% for L.
Table A1. Analysis of Covariance table for response p, productivity. All terms are considered.

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Seq_SS</th>
<th>Adj_SS</th>
<th>Adj_MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
<td>19.5139</td>
<td>0.0009</td>
<td>0.0009</td>
<td>0.01</td>
<td>0.927</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>9.9768</td>
<td>0.3090</td>
<td>0.3090</td>
<td>3.47</td>
<td>0.159</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0.1353</td>
<td>0.6447</td>
<td>0.6447</td>
<td>7.25</td>
<td>0.074</td>
</tr>
<tr>
<td>FL</td>
<td>1</td>
<td>4.5655</td>
<td>1.2147</td>
<td>1.2147</td>
<td>13.66</td>
<td>0.034</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0.3220</td>
<td>0.3220</td>
<td>0.3220</td>
<td>3.62</td>
<td>0.153</td>
</tr>
<tr>
<td>Error</td>
<td>3</td>
<td>0.2668</td>
<td>0.2668</td>
<td>0.0889</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>34.7803</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A2. Analysis of Covariance table for response q, quality index. All terms are considered.

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Seq_SS *10^4</th>
<th>Adj_SS *10^4</th>
<th>Adj_MS *10^4</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
<td>576.90</td>
<td>14.30</td>
<td>14.30</td>
<td>1.35</td>
<td>0.330</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>63.98</td>
<td>85.05</td>
<td>85.05</td>
<td>8.01</td>
<td>0.066</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>5.15</td>
<td>1.67</td>
<td>1.67</td>
<td>0.16</td>
<td>0.719</td>
</tr>
<tr>
<td>FL</td>
<td>1</td>
<td>18.08</td>
<td>27.82</td>
<td>27.82</td>
<td>2.62</td>
<td>0.204</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>135.71</td>
<td>135.71</td>
<td>135.71</td>
<td>12.78</td>
<td>0.037</td>
</tr>
<tr>
<td>Error</td>
<td>3</td>
<td>31.85</td>
<td>10.62</td>
<td>10.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>831.66</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A3. Analysis of Covariance table for response p, productivity. Only terms potentially significant are considered.

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Seq_SS</th>
<th>Adj_SS</th>
<th>Adj_MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>4.5083</td>
<td>0.3276</td>
<td>0.3276</td>
<td>4.90</td>
<td>0.091</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3.5726</td>
<td>0.6438</td>
<td>0.6438</td>
<td>9.62</td>
<td>0.036</td>
</tr>
<tr>
<td>FL</td>
<td>1</td>
<td>25.5726</td>
<td>1.2397</td>
<td>1.2397</td>
<td>18.52</td>
<td>0.013</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0.8591</td>
<td>0.8591</td>
<td>0.8591</td>
<td>12.84</td>
<td>0.023</td>
</tr>
<tr>
<td>Error</td>
<td>4</td>
<td>0.2677</td>
<td>0.2677</td>
<td>0.0669</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>34.780</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Analysis of Covariance -- with all terms for quality index (Table A2) -- signals F and L effects as significant; S and C effects appear not to be significant, while FL is doubtful. Eliminating S and C, the new analysis (Table A4) confirms that only F and L effects are significant, with a confidence limit superior to 99% and 95% respectively. FL is less than 90% significant.

For details on the use of Analysis of Covariance see (for instance) Mason et al. (1989).

Table A4. Analysis of Covariance table for response q, quality index. Only terms potentially significant are considered.

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Seq_SS *10^4</th>
<th>Adj_SS *10^4</th>
<th>Adj_MS *10^4</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>6.70</td>
<td>73.90</td>
<td>73.90</td>
<td>7.78</td>
<td>0.038</td>
</tr>
<tr>
<td>FL</td>
<td>1</td>
<td>587.29</td>
<td>21.00</td>
<td>21.00</td>
<td>2.21</td>
<td>0.197</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>190.20</td>
<td>190.20</td>
<td>190.20</td>
<td>20.03</td>
<td>0.007</td>
</tr>
<tr>
<td>Error</td>
<td>5</td>
<td>47.48</td>
<td>47.48</td>
<td>9.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>831.66</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>