School of Computer Applications

Ollscoil Chathair Bhaile Átha Cliath
Dublin City University, Glasnevin, Dublin 9, IRELAND.

Improving Professional Software Skills in Industry

Rory O'Connor, Howard Duncan, et al.

CA-0201

Working Paper Series

Improving Professional Software Skills in Industry
A Training Experiment

Rory O'Connor, *Dublin City University, Ireland, roconnor@compapp.dcu.ie*
Howard Duncan, *Dublin City University, Ireland, howard@compapp.dcu.ie*
Gerry Coleman, *Dundalk Institute of Technology, Ireland, gerry.coleman@dkit.ie*
Maurizio Morisio, *Politecnico di Torino, Italy, morisio@polito.it*
Cliona McGowan, *European Software Institute, Spain, cliona@esi.es*
Christophe Mercier, *AFTI, France, christophe.mercier@universite.thomson-csf.com*
Yingxu Wang, *IVF, Sweden, wangyx@enel.ucalgary.ca*

Abstract

With over a decade of Software Process Improvement in large organisations, the awareness of its importance has propagated to Small to Medium Enterprises (SMEs). The most well known models for SPI are primarily suited for large- or medium-sized organisations, but with some tailoring they can provide substantial support for SPI in small organisations. The IPSSI (Improving Professional Software Skills in Europe) project aims to address these issues and provide a process improvement framework for use by individual software engineers working in European SMEs. This paper describes the background and motivation behind the IPSSI framework and describes the IPSSI approach to supporting the individual software engineer and reports on a series of case studies using the IPSSI method.

1. Introduction

Software developers and managers have faced the problem of producing quality software since the beginning of the software age. Many people have studied the software quality problem and have proposed solutions, including better testing, better project planning, better practices, better programming environments and many other factors that potentially affect the development of software. We can categorise these different solutions into two groups: Firstly, solutions that focus on software development as a group effort and secondly, solutions that focus on the individual software developer. Some of the many suggestions that involve groups of software developers include: the Capability Maturity Model, clean room development, software quality assurance groups and formal technical review groups. These organisational level methods help improve the quality of the software, however they may not be enough.

Many user needs in SPI are not fully catered for through existing software process models. In particular, there is an absence of support for process improvement at the individual level, although according to a recent survey of European software organisations 83% of companies believe that SPI is essential for future success and 87% of companies believe that SPI can significantly improve software quality [1]. However time, cost and lack of knowledge are seen as the main barriers to SPI usage. In Europe to date, SPI has focused at the organisational level. As a result a number of issues have been identified in the European software industry at this level [2]:

There is no clearly defined framework at the individual engineer level to enable process improvement
Software engineers work with an increasing array of tools in a range of development environments.
The SME needs a suitable method to improve their software engineers' capability for developing software of higher quality on schedule and to budget.
The SMEs need a set of personal process training material that is suitable to the European software industry.

At the personal level, the support available to the individual software engineers in performing their software engineering activities a matter of concern. This is often referred to as the Personal Software Process (PSP), a term coined by Watts Humphrey [3]. The main purpose of providing process resources at this level is [4]:

- To support the automation of mundane tasks and activities.
- To monitor the personal performance of individual software engineers by reporting on the status of their personal software engineering activities.
- To provide guidance and help to software engineers in improving their individual software processes.

The PSP has been successfully used within organisations already using SPI methods, where the culture of quality and process discipline is strong. However, it has not been so successful in smaller or less disciplined organisations. Some of the issues associated with the PSP are:

- There is a significant training period (excess of two weeks).
- It was developed for an academic environment, thus it is difficult to customise for industrial use.
- Many companies involved in PSP training have failed to implement it in a industrial context.
- Bureaucratic overhead makes it more difficult to sustain in an industrial setting
- Lack of an adequate support tool for data gathering.

1.1 The IPSSI Project

The IPSSI (Improving Professional Software Skills in Europe) project [5] is an ESSI funded project which aims to provide a process improvement framework for use by individual software engineers working in European SMEs. The focus of the project is on improving individual software engineering skills thus generating bottom-up improvement. Companies can experiment with SPI by sending individuals on IPSSI training courses and then monitoring its implementation. By training individuals in this way, costs are reduced. At the heart of the IPSSI initiative is the PIPSI (Process for Improving Programming Skills in Industry) approach, whose aim is to present the techniques in a way that makes them more attractive and more easily used in small and medium-sized organisations and development teams.

PIPSI focuses on two areas of concern to the software engineer – Personal Project Management and Personal Quality Management. Both of these are supported by an estimating process and the use of appropriate metrics for measuring past success in estimating and providing a basis for future planning. Each program worked on is treated as a 'project' by the software engineer, who starts with a planning and estimating phase before starting work on the program. As the work progresses detailed metrics are kept, which are used both to monitor progress and to build up a history. These metrics are personal to the programmer at all times, and are not available to management for productivity assessment or similar purposes. The entire personal process is cyclical, success in one personal project building on success in previous projects. As software engineers become comfortable with the concepts, new and more sophisticated techniques can be introduced to improve both Project Management and Quality Management. PIPSI is built around the following key concepts:

- A Personal Process to be used and adapted by the individual. The three main phases of the process are PreBuild, Build and PostBuild. The Personal Process is the foundation needed to define Personal measurement and to use specific techniques and methods, Personal Project Management and Personal Quality Management.
- Personal Measurement which introduces personal measures (effort, size and defects) to monitor and improve the personal process.
- Personal Project Management which introduces activities and techniques to estimate the size and effort of a project, plan a project, schedule it and track its advancement.
- Personal Quality Management which deals with defect collection and analysis to track the quality of a project. Further, it introduces design and code reviews to improve the quality of a project.

The focus of PIPSI is on bottom-up process improvement. Figure 1 illustrates the three elements of personal software engineering - defining a personal process, personal project management and personal quality management.

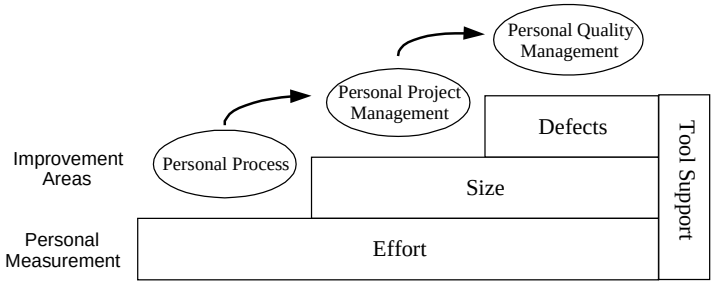


Fig.1 : PIPSI Structure

The entire model is buttressed and controlled through the use of measurement. By collecting data on their own performance, software engineers learn about how they develop software. The measures help them understand the fundamental relationship between size and effort and, through this understanding, enable them to improve their estimating abilities. Furthermore, by gathering data on their defect rates they witness how using practices such as personal code reviews and the use of checklists will allow them to produce higher-quality software products. The measures provide information on performance, information can then lead to process improvement and process improvement can lead to the production of better quality software on time. Finally collecting performance data continuously moves developers from defining their own development process, through managing it to optimising it.

Through PIPSI training, developers complete programming tasks on which they collect increasing quantities of data. Early exercises capture effort measures. Subsequent exercises gather size data whilst the concluding exercises capture defect and quality measures. This is hugely empowering for both the programmer and the organisation as a whole. Programmers are now in a position where they can provide the project manager with achievable deadlines and the project manager can develop more accurate and predictable delivery schedules. The final element of PIPSI is that of personal quality management. As developers complete PIPSI program using exercises, they collect data on the defects injected into those programs. This process illustrates in which development phases they inject and remove defects. Furthermore, the defects are categorised by type thus allowing a causal analysis to be performed which can then lead to defect prevention. PIPSI focuses on proven quality control mechanisms such as design and code reviews which enable developers to remove defects earlier in the development process. This achieves the twin objectives of removing defects at the front end of the development cycle where they are cheaper and easier to fix and, as a corollary, means testing time is more focused as fewer defects are escaping into test.

This process information highlights the developers strengths and weaknesses and empowers them to make the necessary process improvement adjustments. We believe the provision of such a tool will ensure the ‘buy-in’ of training participants and subsequent continued usage of the PIPSI disciplines.

1.2 PIPSI Support Tool

Empirically based process improvement frameworks such as PSP and PIPSI focus on the individual software engineer and require them to record data about time spent programming, the defects they find in their software and size of the software, etc. The PSP as described by Humphrey is a manual process. The engineer records, transfers and analyses he data all on paper forms. After many projects, the engineer accumulates a large paper database of their historical data.

Feedback from PSP training programmes in Ireland [6] has suggested that the absence of a support tool, to simplify the recording and analysis of the data produced is one of the major barriers to continued usage of the methods. Developers tire of recording data on paper forms and eventually usage of the disciplines peters out. There is also corroborating evidence from the USA. For example, [7] and [8] report on experiences of a two year PSP study which questioned the quality of the data recorded. They found that there was significant data quality issues with manual PSP, for example, not all defects were recorded because the overhead in recording was too expensive.

These experiences and observations from the PSP, lead the IPSSI project consortium to consider designing an automated, empirically based personal process improvement tool to support the data collection and analysis requirements of the PIPSI approach to process improvement. Therefore as part of the IPSSI project, a tool set, which consists of data gathering and data analysis tools for use in a web-based environment, has been developed. The main requirements of this tool were that it enables measures to be collected as a simple complement to the development process and provides for analyses of the data collected to provide the developer with important process feedback.

The PIPSI tool was developed in order to support the individual developer using the PIPSI approach and is designed to support two main functions (as illustrated in figure 2): Data Capture - simple and easy recording of data such as time, size and defects, and Data Analysis - automatic analysis of collected data to generate aggregate project data in textual and graphical form. In addition, there were also a number of constraints placed on the development of the tool:

- All data gathered and analysed during a process should be stored in a private database.
- The developer can optionally elect to anonymously submit gathered data to a central database.

- The tool should be platform independent.
- The developer (user) should be able to work in a stand-alone manner, i.e. no network connection required.

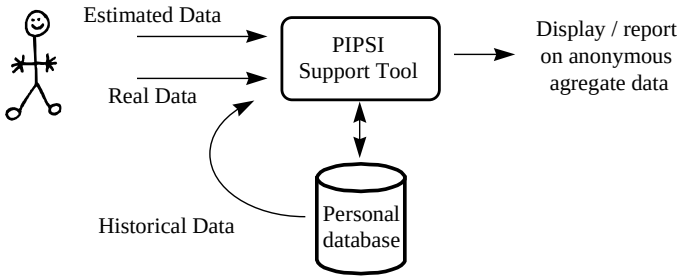


Fig.2 : PIPSI Tool Usage

The PIPSI model follows the three elements of personal software engineering; defining a personal process, personal project management and personal quality management. These are represented in the tool by three main levels:

- Recording basic data such as: size, effort and defects.
- Advanced review techniques and statistical analysis of derived measures including estimated size, real size and defects found in review.
- Expert topics such as design methods, earned value tracking, design errors found, schedule and task planning, productivity and design quality.

The PIPSI support tool was implemented using Active Server Pages which interact with a database system. This provides for a light weight implementation which facilitated the three configurations described above and also provided a flexible approach to system implementation.

The web pages of the system are a code mix of VBScript and standard HTML. The VBScript of the requested web active server page is interpreted by the web server, which generates HTML code which in turn is sent to the clients browser and is displayed like a pure HTML page. The underlying PIPSI data is held in a standard Microsoft Access 2000 database , which is accessed by the Active Server Pages via ODBC connection to the database.

In addition to the design considerations previously described, was the issue of localization of the user interface in terms of the language which is presented on the web browser (user interface). This issues was approached by allowing the user (software developer) to choose at tool startup which language they wished to work in. The text of the user interface is held in a series of files from which the Active Server Pages extract the text to be displayed. Currently the system supports the following languages: English, French, Spanish, Dutch, German, Italian, Swedish, Danish, Finnish, Greek and Portuguese.

2 PIPSI Case Studies

The following sections will provide an overview of the results of a series of selected case studies [15] based on PIPSI training as follows:

- Ireland – undergraduate students, Dundalk Institute of Technology
- Italy – undergraduate students, Politecnico di Torino
- Spain – industrial training, European Software Institute
- Sweden – postgraduate students, Chalmers University

2.1 Ireland

Situated in the North East of Ireland, Dundalk Institute of Technology is one of Ireland's most progressive third level colleges. With student numbers reaching close on 3,000 the college offers courses in the fields of Science, Engineering, Business and Humanities. The Department of Computing and Mathematics, which resides within the School of Science has in the region of 450 full-time students and 150 part-time students. This study details

the results of PIPSI trials. These were staged in an academic environment and involved a group of graduate year Computing students. The trials had two objectives.

- To enable the undergraduates to understand and measure their own software process to effect improvement.
- To provide some quantitative and qualitative feedback about the PIPSI philosophy and the training exercises.

2.1.1 Course Detail

The PIPSI classes were held on three afternoons per week over a 2-week period. 19 students participated at the outset, of which, 16 completed all 5 PIPSI exercises (an 84% completion rate). During each of the 5 programming exercises participants are required to collect successively more detailed data as shown in Table 1.

By following the approach outlined above, participants adhere to the PIPSI model by commencing with effort measures, then progressing to personal project management by relating effort to task size and finally focusing on quality management through understanding defects. Because of the limited number of data points no firm conclusions can be drawn about the disciplines and the approaches. However, the results do provide some promising indications which merit further study.

Exercise	**1**	**2**	**3**	**4**	**5**
Activity	Estimate of time required	Estimate of time required	Estimate of size (LOC)	Estimate of size (LOC)	Estimate of size (LOC)
	Measure Total Time	Measure time by phase	Calculate productivity from programs 1 & 2	Calculate Productivity from Programs 1, 2 and 3	Calculate productivity from programs 1, 2, 3 and 4
		Measure of size on completion (LOC)	Estimate of time based on size estimate and productivity	Estimate of Time based on size estimate and productivity	Estimate of Time based on size estimate and productivity
			Measure time by phase	Measure time by phase	Measure time by phase
			Measure of size on completion (LOC)	Measure defect data by phase	Estimate defects by phase
				Measure of size on completion (LOC)	Measure defect data by phase
					Carry out a code review
					Measure of size on completion (LOC)

Table 1 - PIPSI Data Gathering Requirement

2.1.2 Results

2.1.2.1 Time Distribution

From the first exercise, participants are required to keep track of the time they have taken to complete a particular programming task. A simple development process of "Pre-Build", "Build" and "Post-Build" is followed in all exercises, and the study group recorded their time in each of these areas for programs 2 to 5.

Figure 3 illustrates the breakdown, by phase, of time spent by the group whilst completing the programming exercises.

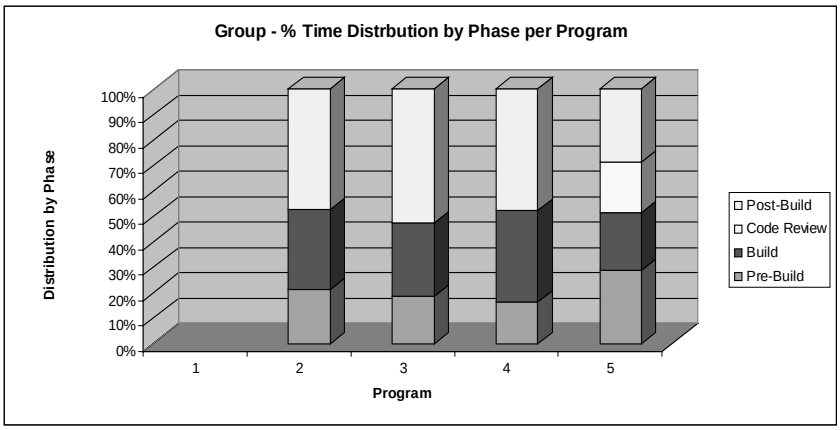


Figure 3 - Group Time Distribution

One of the training objectives was to convince the group of the need to spend more time in the earlier phases of development such as understanding requirements, detailed design and the use of code reviews. Many previous studies have shown how spending a greater proportion of time in the earlier life-cycle phases significantly reduces the amount of time required for testing as the product can be built 'right first time' and less rework and repair is required in test.

Whilst the results above are inconclusive there is a significant improvement in program 5 where code reviews are introduced. From programs 2 to 5 to combined pre-build and build times account for around 50% of development effort. Program 3 has the lowest combined pre-build/build time, which translates into this program having the highest proportion of test time. However, things change in program 5 as participants, on average, spend 20% of their time in code review. This leads to a significant decrease in test time provides encouragement for engaging in code reviews.

Figure 4 shows a student who had some success in reducing test time through increasing effort in the earlier development phases. This student managed to complete all 5 of the PIPSI exercises.

In programs 2 - 4 this student steadily increased the time he spent in design from 29% in program 2 to 40% in program 4. At the same time the proportion of time spent in test has reduced from 57% in program 2 to 40% in program 3. However, the major gain appears in program 5. In this program the student spent almost 70% of his time in pre-build activities. This translated into a much reduced build time. Also, he spent over 15% of his time in code review. All told this reduced post-build (compile and test) time to a mere 6% of development time.
As removing defects as early as possible is a proven way of improving overall program quality this outcome can be seen as a major improvement in the student's development process. Program 5 also involved the smallest time spent in compile (3.5% of total time). This coupled with the reduced time in test may indicate that the student has been successful at removing defects earlier in the development process.

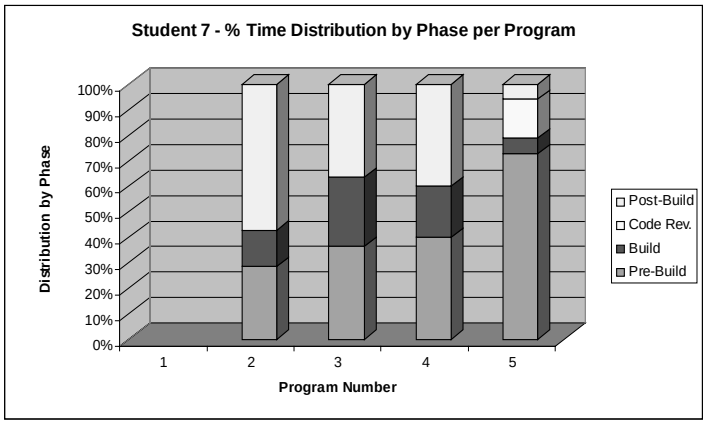


Figure 4 - Student 7 Time Distribution (Programs 2 – 5)

2.1.2.2 Time Estimation

The exercises used for the trials were deliberately short in order to allow for the maximum amount of data to be collected during the training period. Because the exercises were quite small many of the estimating errors are quite large. Several authors have also found that small tasks can generate significant percentage errors. However, over time, when the disciplines have been applied to much larger tasks and sufficient historical data has been gathered, then the error percentages can be reduced. Figure 3 shows the time estimation error for the study group for programs 1 to 5.

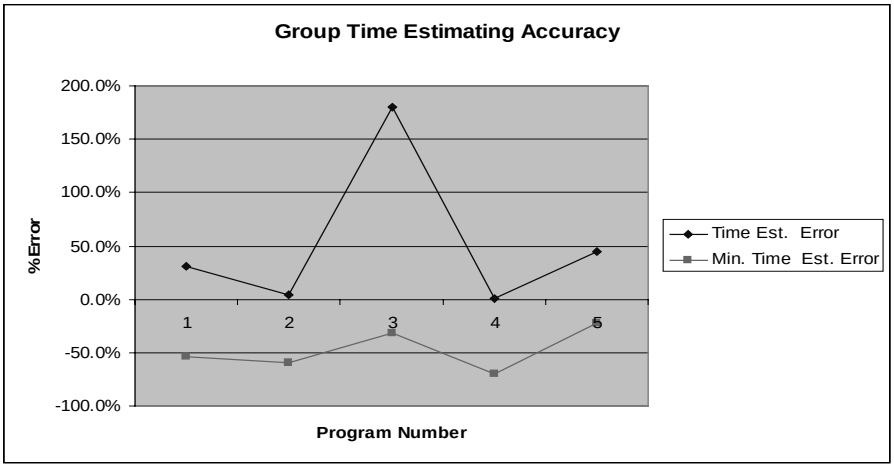


Figure 5 - Group Estimation Accuracy

Firstly, it's interesting to note how the minimum estimation error line tracks the average estimation error line indicating that the students found the same level of difficulty with each program. Whilst the average estimation error line shows that all students underestimated the time required for each program, the average error remains with the +50% range. However, some of the averages are distorted by some very large outliers, particularly in the case of program 3, where 2 students underestimated the time required by more than 1000%.

Student 13 (Figure 6) has managed to exercise some control over time estimation. Their figures show a 50% underestimate for programs 1-3 whilst programs 4 and 5 show major improvement in estimating ability.

In programs 3, 4 and 5 size and productivity measures are used to derive time estimates so if the student collects subsequent figures this would indicate whether the improvement in time estimating is due to this approach.

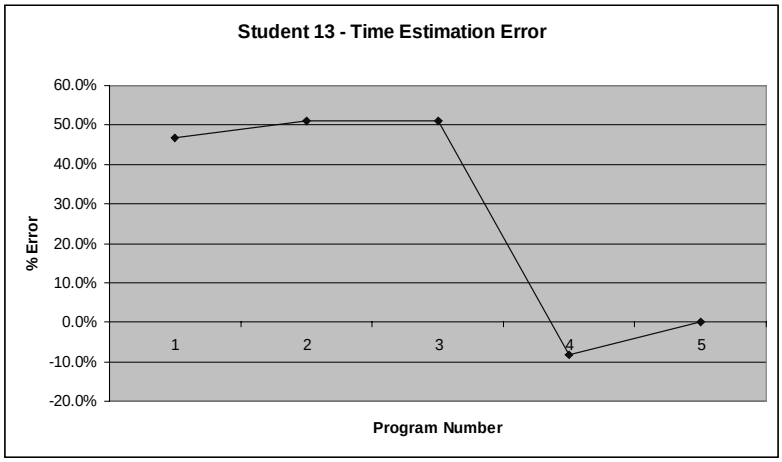


Figure 6 - Student 13 - Time Estimating Accuracy

2.1.2.3 Size Estimation

For programs 3 to 5 for which size data is estimated, the average estimation error for the class ranged from 6.4% in program 3 to 19.3% in program 5 (see figure 7).

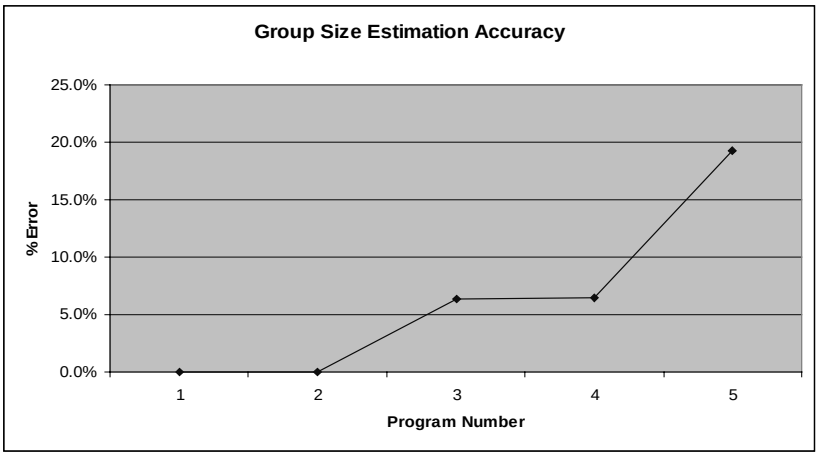


Figure 7 - Group - Size Estimating Accuracy

Whilst on the surface, the accuracy appears to be deteriorating the figures are distorted by some outlying underestimates. If we look at the size estimation error coupled with the standard deviation for each of the programs we get the figures in Table 2:

	Program 3	**Program 4**	**Program 5**
Size Estimation Error %	6.4	6.5	19.3
Standard Deviation	68.5	64.6	17.5

Table 2 – Size Estimation Vs. Standard Deviation

The drop in the standard deviation in program 5 may have more significance than the headline figure of an almost 20% size estimation error. The small average error in programs 3 and 4 hides a wide range of actual error from –61.4% to 231.3% in program 3 to a range of –61.4% to 233.3%. The size estimation error range in program 5 is –12.5% to 47.1%. These figures indicate a much greater clustering around the mean error figure of 19.3% and suggest that all of the participants have a greater understanding of size estimation. Obviously no firm conclusions can be drawn in this regard but should be monitored in future programs.

If we go on to compare program size with development time we can see an interesting relationship between the size of a program and the time taken to develop it. This evidence is shown in figure 6.

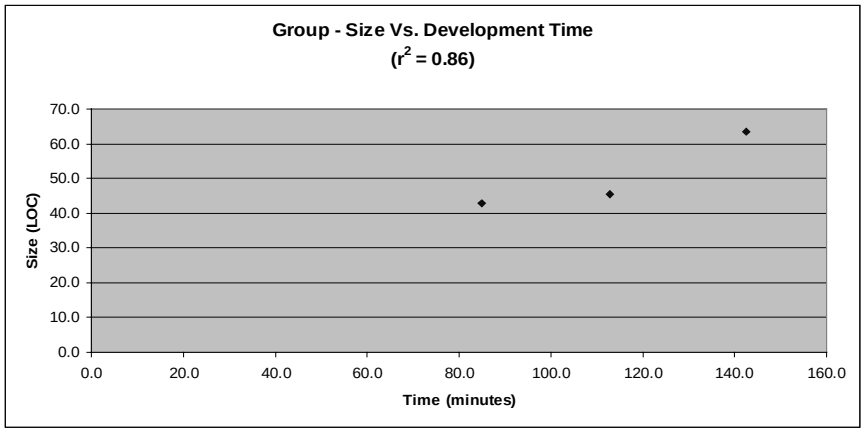


Figure 6 - Group - Size Vs. Development Time

With a correlation coefficient (r) of 0.93 and an r2 value of 0.86 a strong relationship between size and development time is indicated in the figures. Student 1's equivalent data (Figure 7) shows a very strong relationship between size and development time.

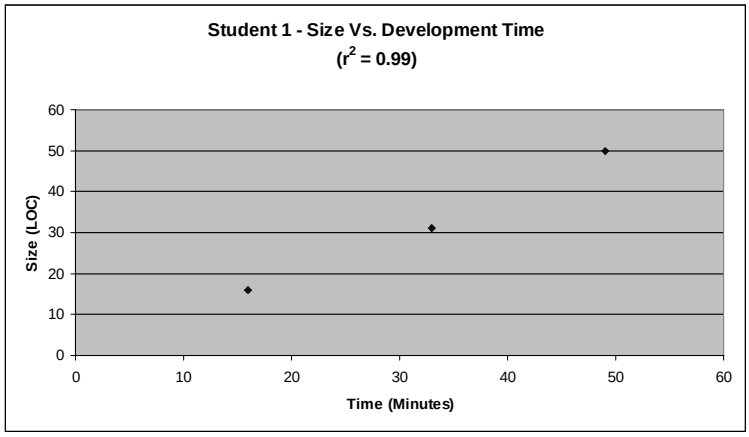


Figure 7 - Student 1 - Size Vs. Development Time

2.1.2.4 Defect Analysis

During PIPSI training, participants collect defect data in programs 4 and 5. Program 5 has an extra dimension in that participants are required to carry out a code review between the Build and the Post-Build stages.

The objective is to illustrate the benefits of code review as a defect detection technique. During the training period at DKIT this seems to have been very effective as Table 3 shows. The defect figures are represented graphically in figure 8.

	Total Defects/KLOC	**Post-Build Defects/KLOC**	**Yield %**
Program 4	240.7	187.6	21.0
Program 5	235.0	76.9	56.8

Table 3 – Group Average Defect Rates (Programs 4 and 5)

Significantly, whilst the average defect injection rate remained almost constant at 241 defects per KLOC for program 4 and 235 defects per KLOC for program 5 the ratio of defects found in the Post-Build has more than halved from a ratio of 187 defects per KLOC in program 4 to 77 per KLOC in program 5. This is reflected in improved process yield (percentage of total defects removed before entry to Post-Build), which has risen from 21% in program 4 to 57% in program 5. It may also be a major reason for the reduction in time spent in post-build by the group in program 5 (see Figure 1).

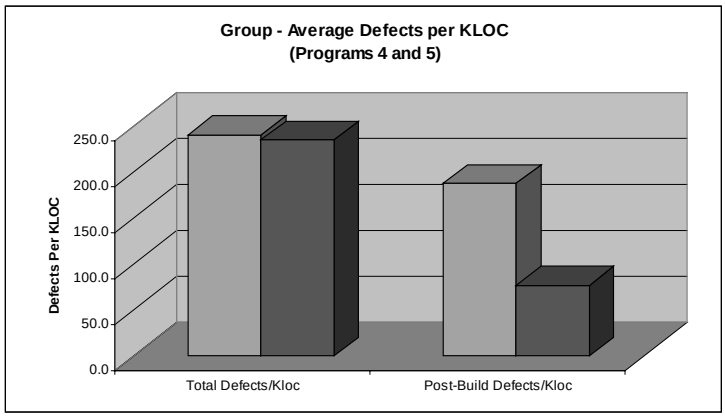


Figure 8 – Group Defect Rates

2.1.3 Summary

Whilst the data above does not provide conclusive proof of the benefits of using PIPSI there are many encouraging signs. Though time estimation has not improved during this instance of PIPSI training the practice of estimating time does possess particular difficulties.

As noted earlier, short exercises can generate quite large estimating errors, as participants tend to overcompensate for previous errors. Also, small tasks will always produce significant percentage errors, e.g. a 2-minute underestimate in a 20-minute task is a 10% error. It is expected that when applied to larger tasks and when size and productivity data are introduced that time estimates will be reduced.

The data do, however, show improvements in size estimating and particularly in defect management. Again, the number of data points is insufficient to supply any definitive proof but the signs are encouraging. Indeed, the data should act as a momentum to participants to continue to use and monitor the PIPSI approaches to determine how they work for them in the longer-term.

There were also some qualitative benefits from the study. Many of the students commented on how they had not thought about programming in this way. A number expressed how previously, they were unaware of the proportions of time they were spending in the various development phases and furthermore had not taken product size into account.
For those who followed them most assiduously, code reviews provided the major benefit and, after the course, several developed a working checklist of potential defect causes to assist with their reviews.

Finally, all expressed the view that the approaches learned through PIPSI would help them in their future programming activities.

2.2 Spain

ESI Training Services facilitates the development, delivery and dissemination of quality Software Process Improvement (SPI) training to software engineers working in a software-intensive environment. ESI's broad training portfolio allows our customers to progress from an introduction to SPI topics to more advanced specialised courses that deal with the diagnosis and implementation of SPI-based programmes. ESI Training Services caters for the software engineer working as a programmer, systems analyst, SPI champion, project leader or manager of a team of engineers. The range of our training products and services is developed in line with ESI's technical programme and our work in emerging, cutting-edge technologies. To support the transition of the theoretical aspects of SPI into actual implementation in the work environment, ESI calls on real project experience at industry level, case study examples and ESI's Best Practice Repository, the most comprehensive database of its kind in Europe. As part of these training services, ESI delivered PIPSI for Managers and PIPSI for Practitioners courses in June 2000.

The course was delivered in three days, the first one, PIPSI for Managers was delivered, and the rest of the days the other one. Eight people attended to the course from four different companies and one university. Next table shows a brief summary of the demographics of the companies that attended to the course.

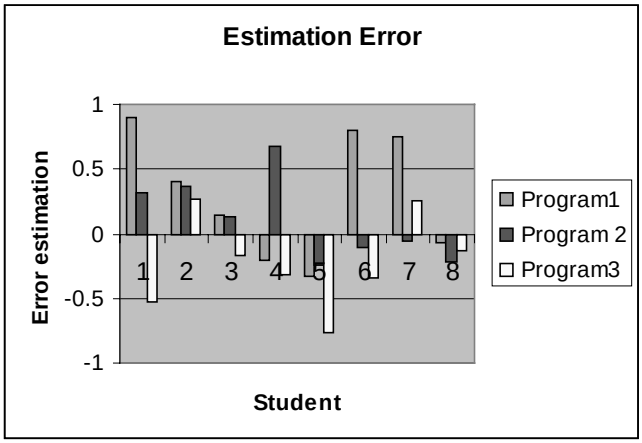
Company	Size of the company	Size of the team	People who attend	Application domain
1	+ 500	11-20	3	Industry
2	+ 500	1-5	1	Banking
3	20-100	1-5	2	Manufacturing
4	500	+21	1	

The objectives of this course were:

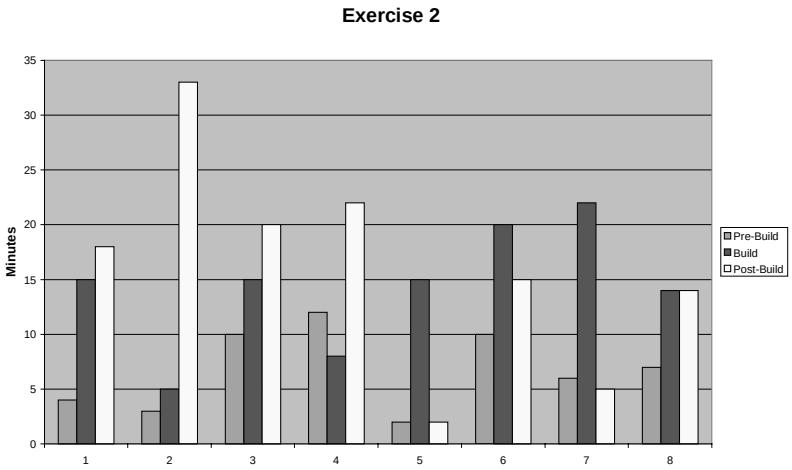
- To understand the need of having a disciplined process.
- To be able to define the own personal process, and improve it in the basis of some measures and analysis.
- To explain the areas of personal software management and quality software management.

2.2.1 Results

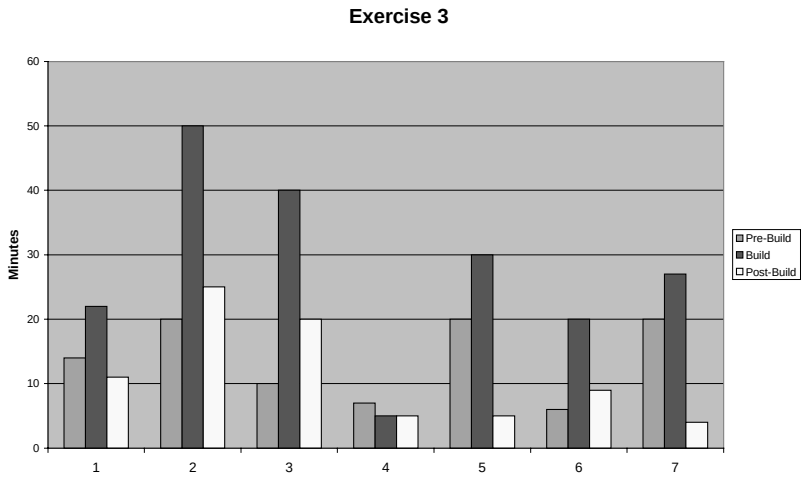
During the courses the students just collected time data, so all the results, that are going to be analysed now, are referring to the time. It is necessary to have in mind that not all the students were software developers, around the half of them did not dedicate their time to develop programs, for example two of them were project leaders and they did not code around ten years ago. For that reason some of the results are not very significant. The following graphic presents the time estimation error between estimate and actual. Although it is not possible to obtain conclusion for all the students, for some of them (for example 2,3,4,8), it can be observed a trend to decrease the error between estimate and actual.



Another analysis that can be done is the time spent in each phase. The following graphics represent the time spent in the exercise 2 and 3 per each phase. It can be observed that during the exercise 2, in the most of the cases, the post-build phase is the phase in which the student spent more time. The post-build phase is the phase where the student looked their defects (compile and test, usually).



During the exercise 3, the time spent in the post-build phase is decreased in the most of the cases. In this exercise 3, the student spent more time in the build phase. The reason for this is that during this exercise some techniques were introduced to help the student to control their defects before post-build phase.



2.2.2 Conclusions

In this section, it is convenient to separate two point of views: teacher and student.

The conclusions of the teacher were:

- The materials are very useful to introduce to the students the concepts of Software Process Improvement (SPI) and to explain in a more detailed way the PIPSI Philosophy, PIPSI elements, the techniques which are used in PIPSI, etc.
- The audience for the course should be clear, this means that the audience for PIPSI for Managers is managers or project leaders, or quality managers and the audience of PIPSI for practitioners is software developers. To maintain these two different audiences will facilitate the delivering of the two courses, because for software managers it is not very useful to know for example how to carry out a code review, but they need to know the benefits of implementing PIPSI. And for software developers could not be very useful to know the benefits that their company is going to obtain, but it is very interesting to know how to improve the work that they do, for example how to deliver a product without defects on time.

The comments of the students are:

- The concepts of the course are very interesting and innovative for them.
- The exercises are very useful to understand and practice the concepts explained during the lectures.
- They found difficulties for estimating the time and the size. Some of them do not have experience in programming, so although they estimated their time and size, they had an error estimation very high. The people, who usually code, had problems because at least in three first exercises they had no data to take as a basis.
- The time assigned to develop the exercises is very short.

Although the concepts are very interesting, they saw several problems to introduce the explained techniques in their daily work. Probably this is caused because they do not have in their companies any mechanism for collecting data. However, the students understood the need to collect the data and analyse them to improve their own processes.

The identified improvements are:

- Improve the requirements of the exercises, this means to clarify the exercise statements adding more examples and more test cases.
- Clarify the audience for the courses, and change the material of the courses according to this.
- Change the agenda of the courses for assigning a bit more time to the exercises.
- Little changes in the courses materials.

2.3 Sweden

This case study presents EHPT's experience in IPSSI training. In this case study, EHPT's approach, achievement and lessons learnt in training of personal software engineering processes are reported. A particular focus is put on software project management and defect management.

The engineers and managers at EHPT have perceived IPSSI personal software process as a practical and useful technology. They thought the EC project can result in good outcomes, particularly on the following:

- Project size estimation
- Project effort estimation
- Project time recording and analysis
- Defect density analysis
- Defect frequency analysis
- Defect removal yield analysis

During the training and practices, the following IPSSI processes have been found most useful in practice:

- Design review
- Schedule estimation
- Personal planning in software development
- Program size estimation

The demographic profile is as follows:

Primary business sector:		Telecommunication software and billing systems
Location:		Gothenburg, Sweden
Size of organisation:		~ 1000 employees world-wide
Annual turn over:	~	200 Million USD
Type of organisation:		A joint venture of Ericsson and HP

A number of software engineering standards and models have been developed in recent decades, such as CMM, ISO 9001, Trillium, BOOTSTRAP, ISO/IEC 12207, ISO/IEC TR 15504 (SPICE), and SEPRM. Software engineering processes can be classified at organisational, project, and personal levels hierarchically according to users of the processes. A personal software process model was first developed by Watts Humphrey known as PSP, which is designed for individual programmers and software engineers in the environment of small organisations and/or small projects. A counterpart of PSP developed in Europe is the IPSSI personal process model.

The IPSSI (Improving Professional Software Skills in Industry) project is funded by the European Commission through the ESSI programme in the framework of ESPRIT. The project aims to provide a process improvement framework for use of tailored and adapted PSP technology by individual software engineers in small and medium sized enterprises (SMEs).

EHPT is one of the successful software companies in Gothenburg that was jointly founded by Ericsson and HP in 1993. EHPT has actively involved in and supported the training course of IPSSI, which was delivered at Chalmers University, Gothenburg by IVF. EHPT found that the IPSSI technology is in line with it's own software process principles and a source for generating idea's for further improvements. Some of the 'activities' defined in the IPSSI model are already in use at EHPT.

The objectives of this IPSSI case study at EHPT are:

- to report the organisation's experience in personal software process management
- to discuss the issues raised and how they were solved

EHPT involved in the whole procedure of the IPSSI personal software process training and teaching support in collaboration with IVF at Chalmers University. EHPT engineers adopted the following approach to gain knowledge and experience of IPSSI personal processes:

- To select suitable project leaders and engineers
- To participate in the IPSSI training course at Chalmers University
- To support the IPSSI training course given by IVF
- To assess how IPSSI technology could improve EHPT's development process.

2.3.1 Experience and Results

The engineers and managers at EHPT have perceived IPSSI personal software process as a practical and useful technology. They thought the EC project could result in good outcomes, particularly on the following:

- Project size estimation
- Project effort estimation
- Project time recording and analysis
- Defect density analysis
- Defect frequency analysis
- Defect removal yield analysis

During the training and practices, the following IPSSI processes have been thought most useful:

- Design review
- Schedule estimation
- Personal planning in software development
- Program size estimation

The major application areas of IPSSI technology at EHPT in the future are identified as follows:

- New staff training
- Software component provider training
- Organisational project data collection and analysis
- Software project quality assurance

In applying and tailoring the IPSSI processes, EHPT considered that the following factors are significant in determining which personal process(es) are going to be adopted and implemented in a given project:

- Size of project
- Importance of project
- Difficulty of project
- Complexity of project
- Domain knowledge requirement
- Experience requirement
- Special process needed
- Schedule constraints
- Budget constraints
- Other process constraints

Despite the positive support for the IPSSI personal process technology, there are a few open issues need to be addressed in IPSSI research and application:

- Reduce the effort of 'overhead' activities in tracking and measure variables for building experience and improving estimations.
- In a software development organisation, when a software engineer is only responsible for specific roles and processes on a project, what kind of personal process model is suitable?
- Adoption of the personal process is very much dependent on project leaders' perception and agreement. Therefore, organisational and cultural changes would be one of the barrels for implementing the personal processes in the industry.
- How do we interface personal processes with team and organisation processes? How do we keep consistency between IPSSI personal processes and the organisational conventional processes?

This case study has reported the experience, approach, achievement and lessons learnt at EHPT in IPSSI training. Particular focus has been put on personal project management and defects management in software development. IPSSI personal software processes and the supporting tool are perceived as an important technology in software engineering that might found a wide range of applications in the software industry.

2.4 Italy

Polito (Politecnico di Torino) is an Italian technical university founded in 1857. It offers curricula ranging from Architecture and Civil engineering to Mechanical, Nuclear and Aeronautics Engineering. It has 2000 staff, of which about 40% are professors and lecturers. It has two main sites in Turin (Engineering school and Architecture school) and several smaller sites in the Piedmont region. Curricula, traditionally organised over a 5 year period, are being restructured in a 3-years (equivalent to a B.S) and a subsequent 2-years (equivalent to a MS) periods.

We used PIPSI with students of the Telecom engineering degree (still organised as a five years curriculum), attending a software course in their fourth academic year. The course, the fourth and last with software contents in the curriculum, teaches essential notions in object orientation (UML, patterns) and concurrent programming. The previous software courses teach fundamentals of computing, a programming language, data structures and algorithms. There is no course in software engineering (lifecycle and software process, metrics, testing, configuration management, project management, requirements engineering). The students were proposed to attend PIPSI on a voluntary level, as an introduction to software engineering topics, and without any evaluation at the end of it, except a sort of interview on their experience. A minority of the students attended PIPSI that was delivered outside normal teaching hours, in three sessions in three different days.

We summarise here a characterisation of the students.

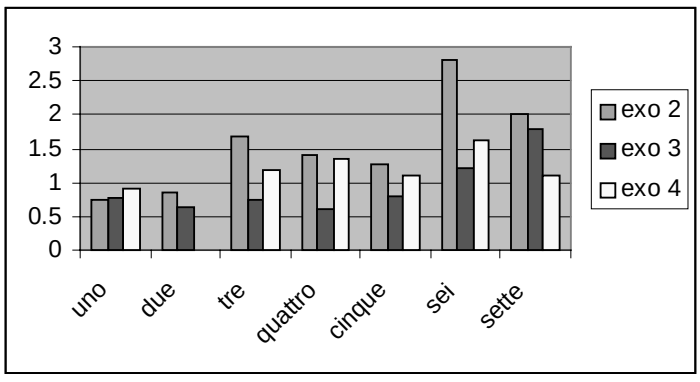
Age	22 to 24
Experience in programming	Fair to good
Experience in industrial programming	Low
Experience in process and metrics	Low to very low
Language used	C++ and C

The course was delivered in three sessions, in three different days, using the training material that composes the PIPSI for practitioners. During the course the students completed three exercises (specifically exercises 2 , 3 and 4 of PIPSI for practitioners). Some of the students did not attend all the training sessions, so some data points are missing. During each exercise, the students logged data, according to the PIPSI guidelines, and specifically estimated and actual effort, size and defects.

2.4.1 Results

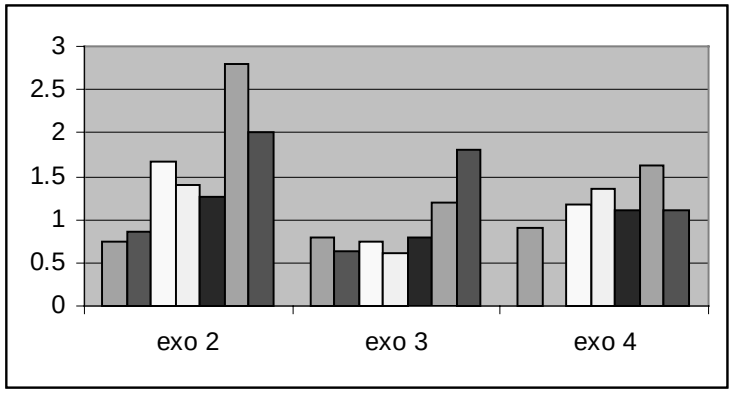
The charts below show the main measures collected during the course.

2.4.1.1 Effort estimation



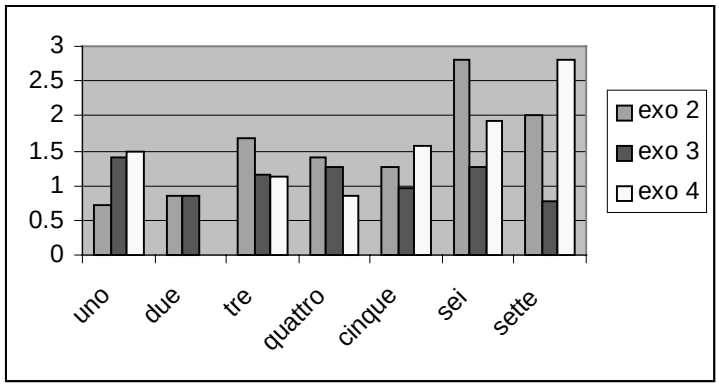
As already mentioned, each student logs, for each exercise, the estimated effort, before starting the exercise, and the actual effort when he finishes it. It is interesting to plot these figures, to check if the accuracy in estimation varies.

The chart above shows, for each student, the ratio actual effort/ estimated effort (an indicator of accuracy in estimations) for each exercise. Although it is not possible to derive general conclusions, for some students (uno, tre, cinque, sette) a trend of improvement is visible.



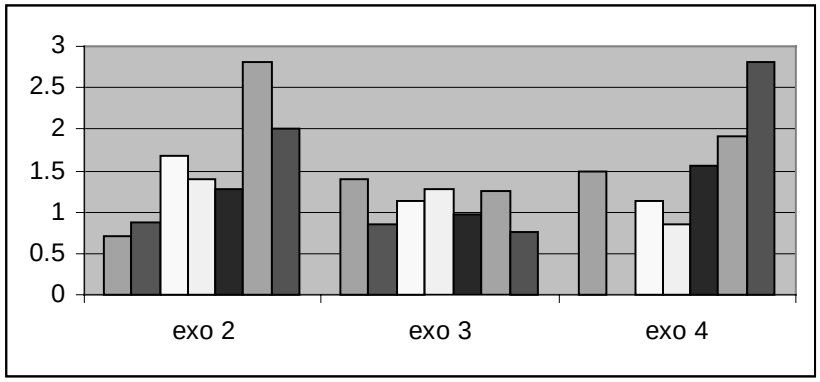
Above we show the estimation errors by exercise. Here the trend towards a reduction in spread is much clearer. We interpret this result as an effect of increased experience in estimation due to the application of PIPSI concepts.

2.4.1.2 Size estimation



In a similar way as for effort, each student logs the estimated size for an exercise before starting it, and computes the actual size at the end. Again, we are interested in checking the trend in size estimation accuracy.

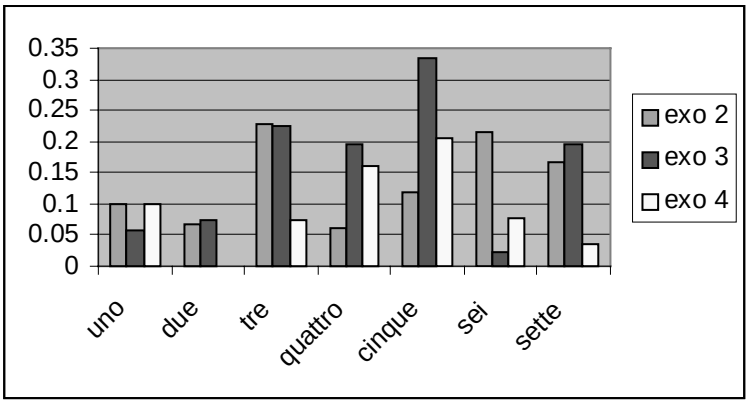
Above we show, for each student, the ratio actual size / estimated size. While in two cases (sei, sette) there is clearly no improvement, in the other cases there is stability or a slight improvement.



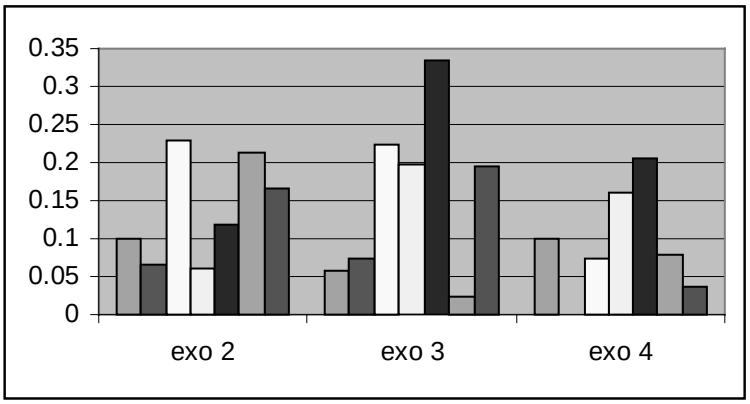
Above, the same size estimation error rate is shown by exercise. The spread is reduced in the second exercise, but not in the third. Overall, there is no evidence of improvement in size estimation. However, exercise 4 requires reuse of code previously written. This could introduce a new factor that disrupts the learning process.

2.4.1.3 Defect density

Students log, for each exercise, the defects that they find and fix. Since they compute also the size of the program, it is possible to compute the defect density. Defect density is a better indicator of quality than the simple count of defects. We are interested in checking if defect density varies with the usage of PIPSI techniques.



Above we show defect density per student, per exercise. We cannot recognise a clear improvement trend.



Above we plot the defect density per exercise, per student. Here there is a slight trend in reduction of spread.

2.4.2 Conclusions

From the point of view of the teacher, PIPSI was deemed very useful to give to the students an introduction to software engineering concepts and methods. The course was especially suitable for students without any notion of software engineering. On the other hand, the course would not be suitable for students with a deep knowledge of software engineering. The structure of the course, with theory and exercises to apply it in practice, was deemed very useful.

From the point of view of the students, the main comments were:

- Interest for the exposition to new concepts
- Appreciation for the practical application of concepts, with a nearly one to one correspondence between theory and practice

- Difficulty in agreeing on the interest of estimation. This depends both on the small time required for exercises, and in the lack of industrial experience. In fact, students with some form of experience of work in real settings were more interested in estimation.
- Difficulty in agreeing on the lifecycle and the three phases (pre build, build, post build). Again, this was due to the small time required by exercises. However, the students agreed that the problem was a problem of scale, and not conceptual.

Overall, PIPSI was considered as well suited for a fast and effective introduction to essential and practical software engineering. It will be considered to become a standard part of the curriculum.

2.5 France

AFTI (Association pour la Formation aux techniques industrielles) is an association created for developments of computer science training. Since 1990, AFTI delivers a Master course with a state diploma (one hundred diplomas delivered every year), in a sandwich course organisation, integrating new recruits from THOMSON, ALCATEL, Aerospatiale, Hewlett Packard France, IBM France, France TELECOM, etc The Master curriculum includes software project simulation called OBA project. The software to be developed is a real time embedded software for a Car Computer. The Driving Assistance System shall improve the safety of a vehicle, especially for long trips on motorways. This includes : speed regulation, trip information (mean speed, fuel consumption...), help for safety (services check).

OBA main goals are:
- apply methods and tools, learned during the course, through a complete software development , with the requirements of the DOD 2167A and using a defined Software Case Tool.
- follow the CMM requirements (partly level 2 and 3), including project management, quality management and configuration management. The Personal improvement Process introduced during the Course, is implemented during this project by the apprentices.

Through the pedagogic coaching, students point out lessons learned concerning different parts involved in software development (project, quality or software manager, customer, system engineer, secretary...) At last, students are trained to the importance of communication: with daily progress meetings, contractual and peer reviews, notified requests to the customer.

The development is split into different phases validated with a formal review. The pedagogic team plays different roles: customer, technical and methodology expert. Students are organised in groups of five or six persons. Each member of the group holds a specific responsibility , changing at end of phase (quality manager, project manager , configuration manager, developer).

During the Master curriculum, students follow an adapted PIPSI course. During the OBA project, the students apply the PIPSI concept on the different phases of the project : requirement analysis, design, coding, unitary tests, acceptation). PIPSI is applied for all the activities made by each students like writing documentation, uses cases, test scenario.

The objectives of the OBA case study:
- To be able to define the own personal process for all software phases (Requirement analysis, Design, Coding, Unitary tests)
- To be able to define measures for these activities.
- To be able to analyse and improve their own personal process

2.5.1 Results

A class of 18 students
The AFTI case study has been performed by 3 groups of 6 students.

Heterogeneous groups

These students have only 7 month practice in company. Each student have it's own competence in different subjects. For example one student has an experience in configuration management but has no experience in object programming.

PIPSI introduced into the methodology of development

PIPSI is used during each software development phases and also for the documentation writing. At the beginning of each phase, software manager with the coach define how to implement PIPSI during this phase. They define :

- input and output of the phase.
- the personal process to implement. This process is based on the PIPSI framework : pre-build, build, post build.
- the size measure. This size measure should be adapted to the activities.
- Error measurement.

Peer reviews added to the basic PIPSI phase

In the OBA project, peer reviews belong to the initial planning. Peer reviews are added to the basic PIPSI standard framework.

PIPSI tool implemented on the intranet

TIPSI tool is set on the private ETGL network. Tool is accessible from all students computer. Students configure the tool for each phase.

PIPSI implementation

This part describes for each phase how PIPSI is implemented.

Requirement analysis:

- PIPSI is applied for use case writing
- Input : System Segment Specification
- Output : Software Requirement Specification (use case)
- Process defined.
- Pre-build : System Segment Specification analysis
- Build : writing the use case
- Use case peer review.
- Post build : correction.
- Measurement size : use case step.

Design

PIPSI is applied for class writing and detailed design of the methods.
Input : Software Requirement Specification, Software Design Document (architecture design)
Output : Software Design Document (complete)
Process defined.
Pre-build : Software Requirement Specification and architecture design analysis
Build : writing detailed design of each methods.
Detailed design peer review.
Post build : correction.
Measurement size : the number of method of the class. For each method, a weight is estimated.

Coding

PIPSI is applied for code writing.
Input : Software Design Document
Output : code
Process defined.
Pre-build : Software Design Document analysis.
Build : writing code.
code peer review.

Post build : compilation and correction.
Measurement size : the number of line of code.
Defect measurement : syntax error, logical error …

Unitary test
PIPSI is applied for unitary test under Attol tool.
Input : code
Output : code and unit test result.
Process defined.
Pre-build : unitary test definition.
Build : writing unitary test.
Post build : unitary test execution
Measurement size : the number of line of code write for unitary test.
Defect measurement : syntax error, logical error..

Qualification test definition
PIPSI is applied for acceptance test writing.
Input : Software Requirement Specification
Output : Software Test Description (scenario)
Process defined.
Pre-build : Software Requirement Specification analysis.
Build : acceptance test writing .
Post build : acceptance test execution.
Measurement size : the number step on the acceptance test scenario.

Document writing :
PIPSI is applied for writing Development Plan.
Input : Software Development Plan framework.
Output : Software Development Plan.
Process defined.
Pre-build : Software Development Plan framework analysis
Build : writing the document
Post build : correction after peer reading.
Measurement size : number of paragraph.

2.5.2 Conclusions

PIPSI has been used by the three groups. Due to the project size, each student has applied PIPSI only three or four times in each phase. This number is considered as insufficient to see progress from the PIPSI method. But the work made in defining the PIPSI process for a phase is interesting by itself to help the student to organize this work. Some difficulties are encountered by the students for time measurement and size measurement. This task is considered by few students as an additional work despite tools. The tools is useful for time measurement and collect data. It helps student to perform analysis of their data.

In some case, the PIPSI result gives measurement of the capacity of learning a tool or a method. For example, the Attol tool is not well known by the students. The error between the estimate time and the realized time decreases at each project. This error should be consider as the learning effort.

PIPSI has an impact on the software manager role. It needs to define activities more precisely for each developer. Developers have more responsibilities on their own work, they need to follow their own process. They are able to give a best estimation of the work to be done.

PIPSI is now a part of the OBA project. This first experience gives good result of the PIPSI implementation. Some improvements are needed like define defect measurement and a best integration of the tool.

A FEW FEATURES (Mean Values)

Group size	6 persons
System requirements	around 20
Number of source lines	4000
C++ classes developed	40
Time evaluation	1800 hours
Phases	
Requirement analysis	11 days
Design	9 days
Coding and unit tests	11 days
Integration	3 days
Qualification	2 days
Debriefing	1 day
TOTAL	37 days

3. IPSSI Results

This section presents a formal analysis of all regional IPSSI training programmes from training delivered by IPSSI partners in Europe. The sample of the population that is taken into account for developing this analysis is the nineteen students. The reasons for this are:

- Not all the student performed the same number of exercises.
- Not all the data are comparable in an analysis because the students could use any language, any platform, and so on to carry out the exercise.

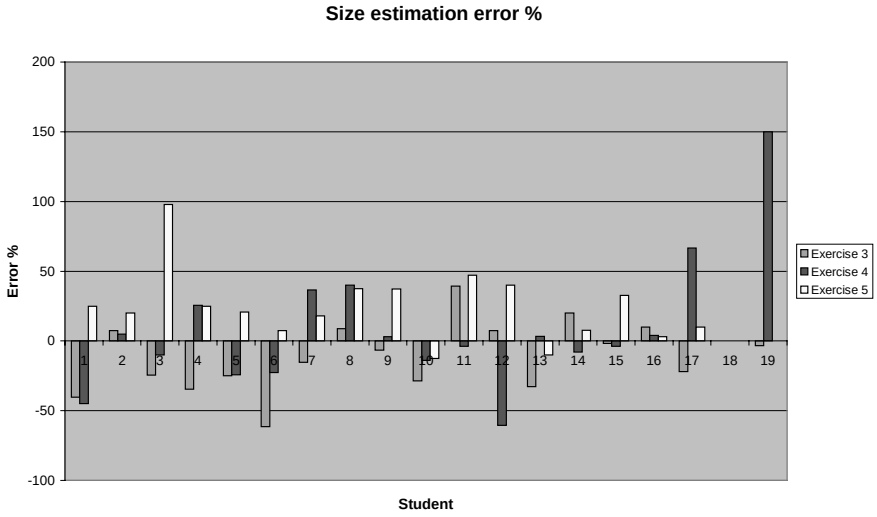
So, this report analyses a sample of the population, and the most of the results are extrapolated to all the students. It has been tried to group any students with same characteristics.
The common characteristics are:

- All the students developed the five exercises
- All the students collected the same data for each exercise:
- Exercise one: time
- Exercise two: time per phase (pre-build, build, post-build)
- Exercise three: time per phase and size (LOC)
- Exercise four: time per phase, size (LOC), defects injected per phase and defects removed per phase
- Exercise five: add a new phase to the process: code review.

The main analysis performed in this report is those referring the size, time and defects.

3.1 Size Analysis

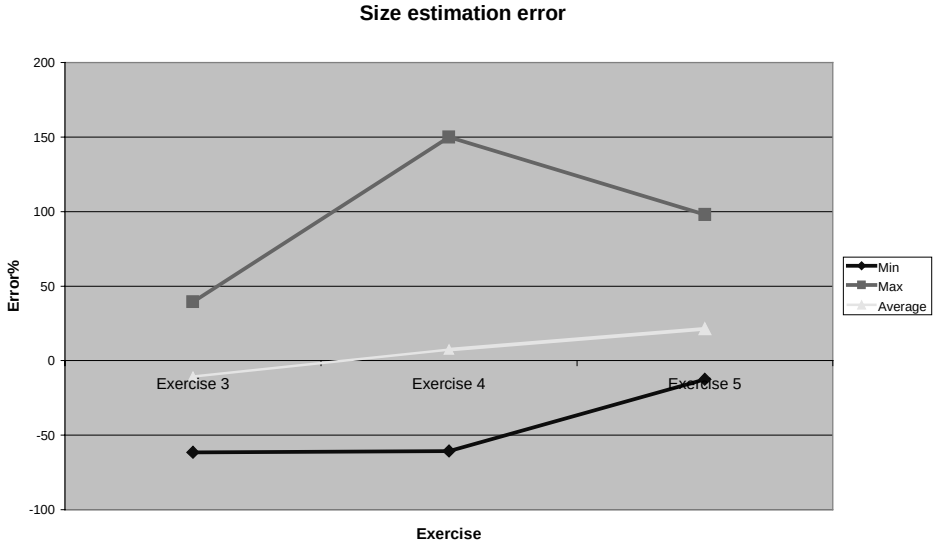
In this section two graphics are going to be shown: one indicating the size error estimation percentage for each student in the last three exercises, and in the other one is shown the min, max and average of this rate for all the students.



Graphic 1

This graphic represents the size estimation error. The size estimation error is an indicator of the accuracy in the size estimation. The important thing of this graphic is that the size estimation error is decresing in each exercise. This is a clear consecuence of the use of the previous collected data to estimate the size of the next exercise, with this method the students achieved to estimate more accurate. There are several examples, the student 19's size estimation error in the last exercise (exercise 5) is 0, although the estimation error for the other students is not 0, this does not imply that they did not improve, because the most of them achieved the decrease of this estimation error, for example student 16, student 10, student 1, and so on.

In the bellow graphic, it is shown three lines which represent the minimum, maximum and average of the size estimation error for all the students in the exercises 3, 4, and 5. The data collected do not allow to perform this analysis in the exercise 1 and 2 because there are not size data.
The important issue is this graphic is that the lines are each time nearer 0. If there would be more exercises, probably it will be seen that the distance between the maximum line and minimum line is smaller, and all the lines will be near 0.



Graphic 2

3.2 Time Analysis

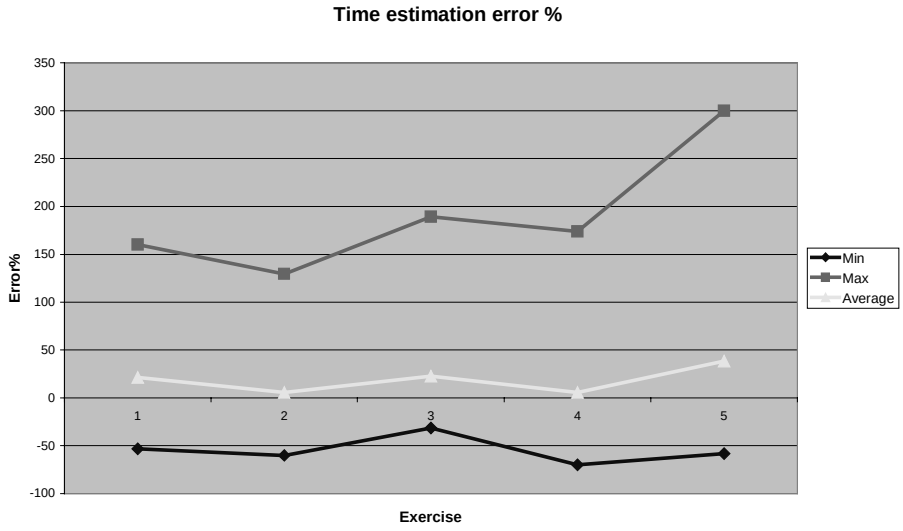
In this section, the analysis will be focused in three different issues:

- Time estimation error

- Time percentage spent per phase in each exercise
- Productivity

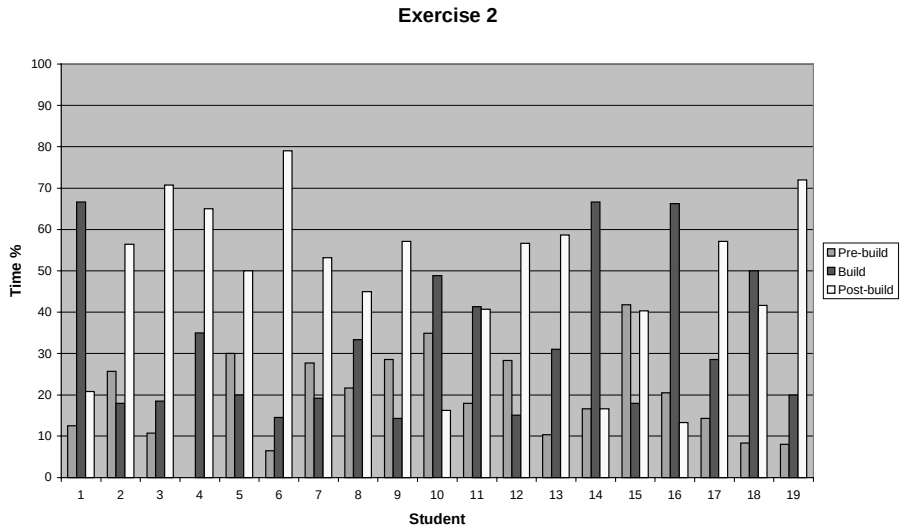
The first aspect is the time estimation error. Like in the size estimation error, the time estimation error is an indicator of the accuracy of the time estimation. The graphic bellow presents three lines: minimum, maximum, and average of the size estimation error for all the students in the exercises 3, 4, and 5

In the graphic below, it is shown three lines which represent the minimum, maximum and average of the time estimation error for all the students in all the exercises. The important thing is to look at the progress of the lines. In this graphic there is not a clear trend on the improvement of the time estimation error. This is because the process used during the exercises is not a stable process, this means that each exercise added some new activities or phases to the previous process, and this causes that it is quite difficult to make accurate estimation because the historical data are not complete. Once a stable process exists the accuracy of the time estimation will be improved.



Graphic 3

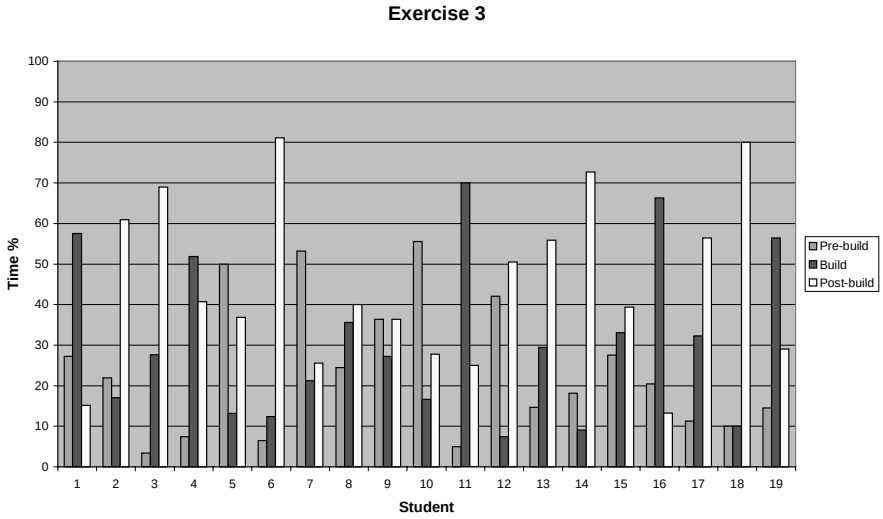
The following graphics represent the percentage of time spent per phase for each student.



Graphic 4

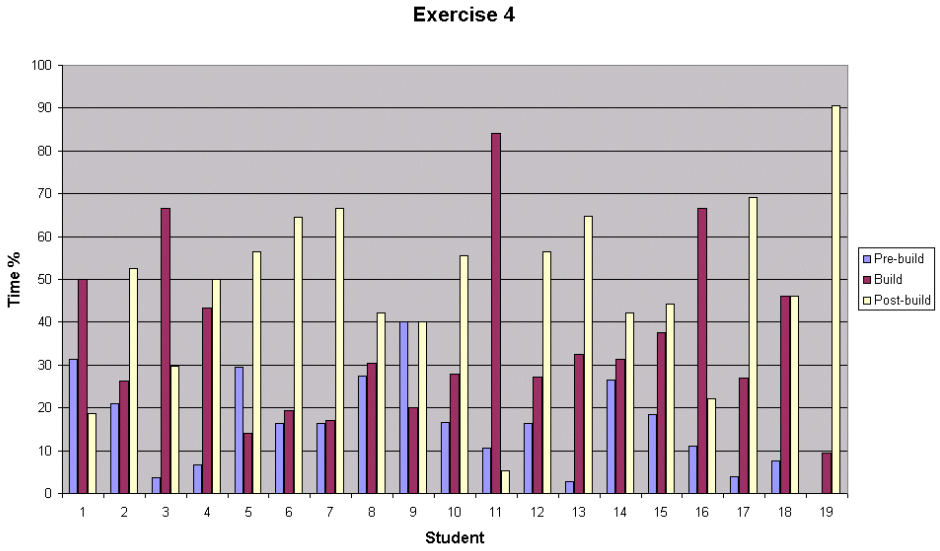
For the exercise 2, the most of the student spent a high percentage in the phase of post-build phase, which is the phase where the defects are removed. The pre-build phase is the phase with the smaller percentage, this implies that the student spent little time in the activities prior to the codification (usually, plan and design). This aspect

probably causes that during the build phase a lot of defects are introduced, and obviously these defects should be removed in the post-build phase and for this reason the percentage in this phase in so high.
Let's go with the exercise three:



Graphic 5

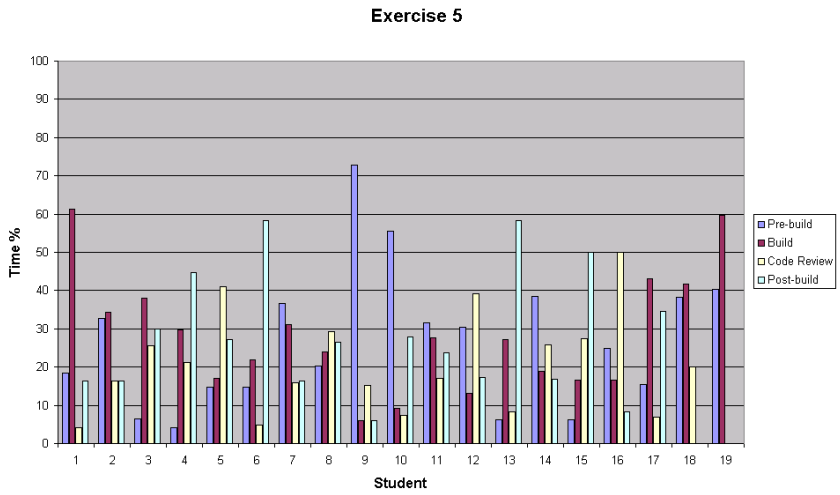
In the exercise 3, the percentage for the post-build phase follows quite high in the most of the cases. But it can be observed too, that the percentage of the time spent in pre-build phase has increased. In the cases where this percentage has increased in a higher way, the percentage for the post-build phase has decreased. For example student 6, 5, 19, and so on.



Graphic 6

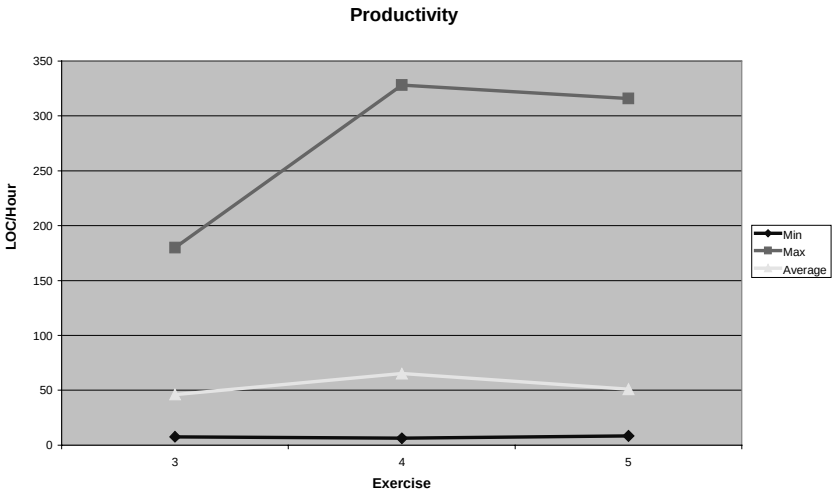
In the exercise 4, the concept of the defect is introduced, and the students should be aware of the defects they introduce. In this exercise, the percentage of the pre-build phase continues increasing, and the percentage of the post-build phase decreases for the most of the student. If the data of the student 19 are observed in detail, it can be seen that the percentage of post-build phase increased very much, this could be a consequence of the lack of the pre-build phase.

Finally, the exercise 5 introduced a new phase: "code review". This phase is introduced before the post-build phase, and helps to find and remove defects before in the development cycle, because to remove the defects as soon as possible results in a saving money and improving the quality of the programs.

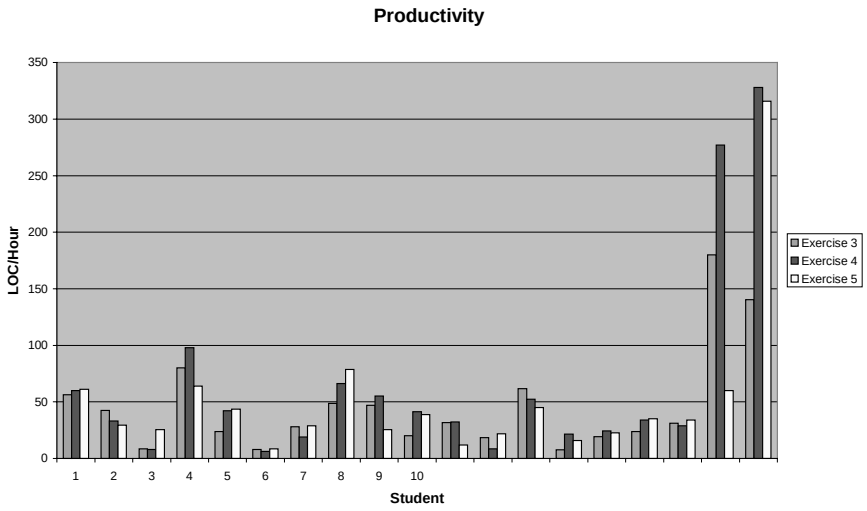


Graphic 7

The last issue referring to the time joins the two sections analysed until now: time and size. The productivity is and indicator of the LOC developed for the student in an hour. The objective of PIPSI is not to improve the productivity; i.e. the objective is not to develop software quicker, but to develop better software products. For that reason, the following graphics do not present high improvements. The first one shows three lines which represent the maximum, minimum and average the productivity for each exercise. The second one shows the productivity of the exercises 3, 4, and 5, for all the students.



Graphic 8



Graphic 9

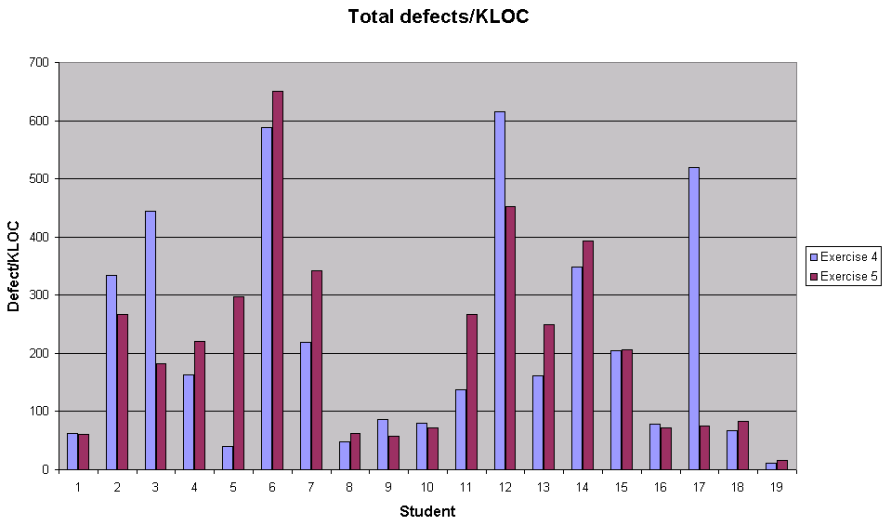
3.2 Defect Analysis

The defect data are collected starting from the exercise 4, so to do the analysis there are only two data points: exercise 4, and exercise 5. But looking at the progression in these exercises, it could be concluded that this progression is going to continue if the students continue applying this defined process.

The graphics that are going to be presented and analysed are:

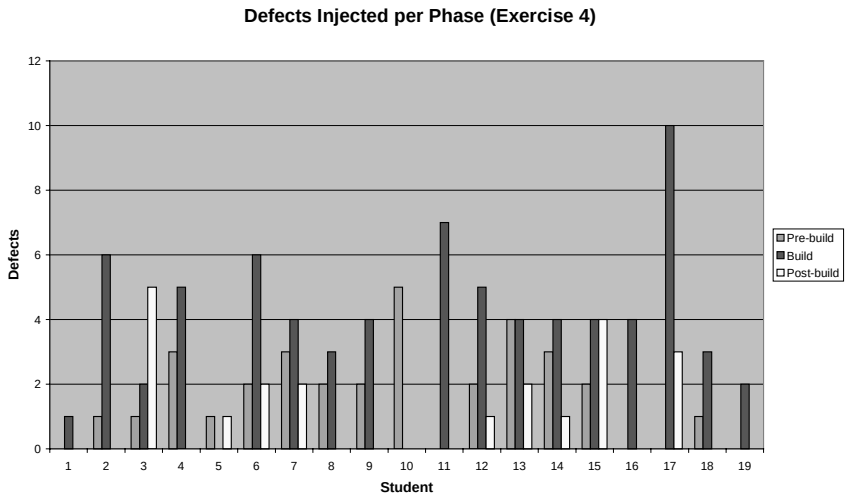
- Total defects/KLOC per exercise and for all the students.
- Defects injected per phase for all the student
- Defects removed per phase for all the student

The first graphic is only for showing that all the students introduced defects and the number of defects did not decrease a lot but this is not the objective, the objective of PIPSI is to find the defects as soon as possible.

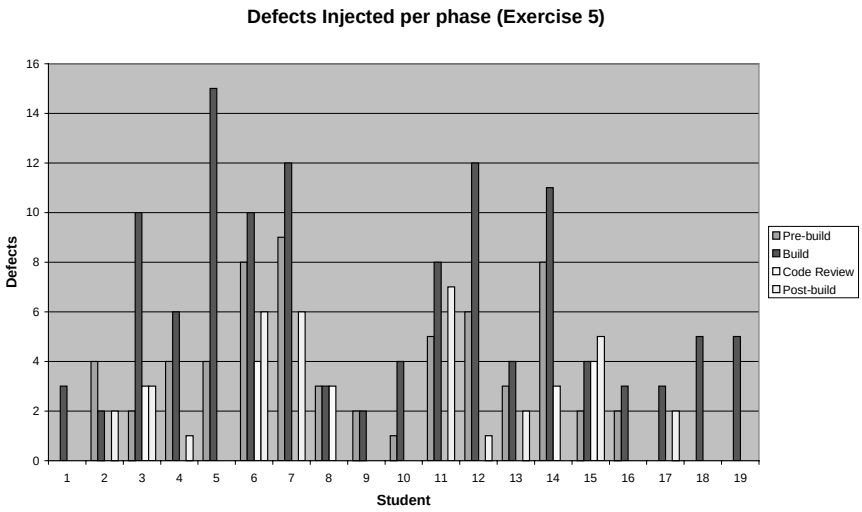


Graphic 10

The following graphics are shown the defects introduced during the different phases in the exercises 4, 5. It could be surprised that in a phase (post-build) where normally defects should be removed, some defects are injected, this is because, when a student corrects a defect, he could be injected another one. For that reason, as in the exercise 4 as in the exercise 5, some defects appear in these two graphics.

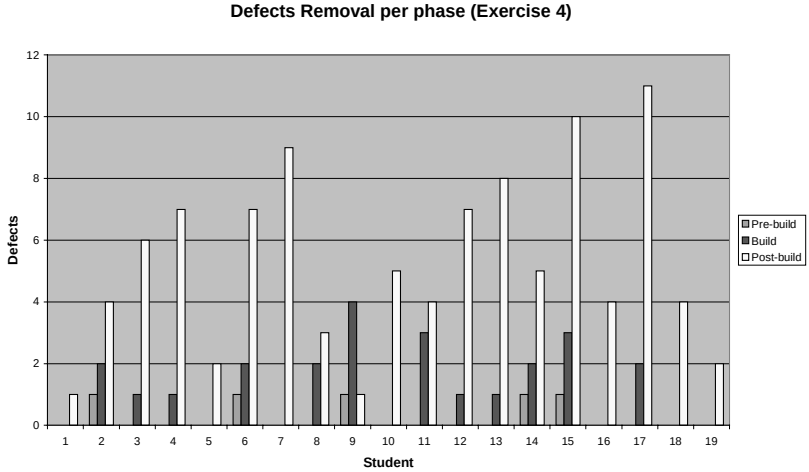


Graphic 11



Graphic 12

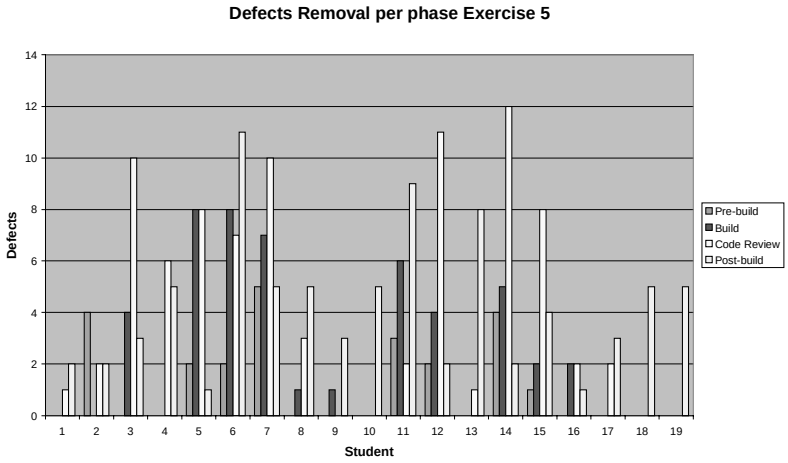
The last graphics are shown the defects removed per phase.



Graphic 13

It can be observed that the most of the students removed their defects in the post-build phase. This is just what PIPSI tries to eliminate, because as later a defect is removed as more expensive is to correct it. For that reason in the exercise five a new phase is added: code review. The objective of this phase is to detect the defects before the post-build phase, and it can be observed in the following graphic that the number of defects removed in the post build phase are decreased.

Some of the student did not include this phase so they did not improve the number of defects removed in the post-build phase.



Graphic 14

3.3 Summary

The most important conclusion is that although some improvements can be observed in the analysis of these data, these improvements can be increased when the students apply these concepts and techniques to their daily work. Having a good database with historical data is very important for obtaining the benefits of applying all the concepts and techniques that are explained in PIPSI courses.

4. Overall Conclusions

The experience of the IPSSI consortium in investigating industry requirements and defining an appropriate method has been worth while and rewarding. While the response from industry to the method developed has been slower than anticipated, the reaction has been positive, and it seems clear that there is a potential market for such methods. For example, there has been considerable interest from managers and academics who have heard about these methods, but little practical follow up.

In view of this, the consortium has decided that the best future for methodologies such as PIPSI lies in making they more widely and easily available. We have therefore decided to make PIPSI available as ‘freeware’, accessible from the IPSSI web page [5]. In this way, it is anticipated that a wider recognition of the method will grow, and individuals and companies will be encouraged to experiment with it.

Based on results to date [10], using a defined, planned, and measured personal process appears to offer many benefits. Assuming that further experience confirms these early results, one would expect methodologies such as PSP and PIPSI top be widely adopted and used. However, from experiments reported in this paper and elsewhere, it is also clear that industrial introduction is not easy.

Some SPI researchers [11] believe that the future of such individual-level SPI methodologies is not as an as-is process for industrial contexts. Rather, researchers, can use it as-is in academic teaching, where they can change context variables. Engineers can use it as a process template to inspire team or group process improvement.
Presuming the demand for skilled software engineers continues to increase, the most appropriate way to introduce SPI may be through the educational system. Rather than having students first learn undisciplined practices and then unlearn them, we should introduce disciplined methods at the beginning of the curriculum.

5. References

[1] "The SPIRE Handbook", Centre for Software Engineering, Ireland, 1998.

[2] Y.Wang, H.Duncan, M.Kartinnen, H.Sjostrom and P.Kokeritz, “IPSSI - A European Methodology on PSP”, Proceedings EuroSPI-99, Finland, 1999.

[3] W.Humphrey, “Introduction to the Personal Software Process”. Addison Wesley, 1997.

[4] S.Zahran, “Software Process Improvement - Practical Guidelines for Business Success”, Addison Wesley, 1998.

[5] http://www.compapp.dcu.ie/ipssi

[6] P.O'Beirne and J.Sanders, “Personal Software Process: Does the PSP Deliver its promise?”, Proceedings of Inspire ‘97, Gothenburg, Sweden, 1997.

[7] A.Disney and P.Johnson, “Investigating data quality problems in the PSP”, Proceedings of the ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering, Florida, USA, 1998.

[8] C.Moore, “Personal Process Improvement for the Differently Disciplined”, Proceedings of the International Conference on Software Engineering, Los Angeles, USA, 1999.

[9] G.Coleman and R.O'Connor, “Power to the Programmer”, In Proceedings of 11th European Software Control and Metrics conference, Germany, 2000

[10] W.Humphrey, “The Personal Software Process: Status and Trends”, IEEE Software, December 2000.

[11] M.Morisio, “Applying the PSP in Industry”, IEEE Software, December 2000.