Comparison of software development life cycles: A multiproject experiment

Article *in* IEE Proceedings - Software · July 2006

DOI: 10.1049/ip-sen:20050061 · Source: IEEE Xplore

3 authors, including:

Darren Dalcher
Lancaster University
178 PUBLICATIONS **875** CITATIONS
SEE PROFILE

Helgi Thorbergsson
University of Iceland
28 PUBLICATIONS **127** CITATIONS
SEE PROFILE

Prepublication version – publisded in IEE Proc.-Softw., Vol. 153, No. 3, June 2006, pp 87-101 - © IEE

Note: In the published version of this paper (in IEE Proc.-Softw., Vol. 153, No. 3, June 2006) bold and italics formatting has been left out when printing numbers. Bold and italics formatting are used to indicate statistical significance. The readability of the published paper is thereby impaired.

Comparison of Software Development Life Cycles: A Multiproject Experiment

Oddur Benediktsson, Darren Dalcher and Helgi Thorbergsson

Oddur Benediktsson
University of Iceland,
Dunhaga 5, IS-107 Reykjavik, Iceland
oddur@hi.is

Darren Dalcher
Middlesex University
Trent Park, Bramley Road
London N14 4YZ, UK
d.dalcher@mdx.ac.uk

Helgi Thorbergsson
University of Iceland
Dunhaga 5, IS-107 Reykjavik, Iceland
thorberg@hi.is

Abstract

A variety of life cycle models for software development are generally available. Many of the variations were composed to overcome problems in the classic waterfall model. However, it is generally difficult to compare and contrast the methods and very little literature is available to guide developers and managers in making choices. Moreover in order to make informed decisions developers require access to real data that compares the different models and the results associated with the adoption of each model. This paper describes an experiment in which fifteen software teams developed comparable software products using four different development approaches (V-model, incremental, evolutionary and Extreme Programming). Extensive measurements were taken to assess the time, quality, size, and development efficiency of each product. The paper presents the experimental data collected and the conclusions related to the choice of method, its impact on the project and the quality of the results.

1. Motivation

Should the Waterfall model be dropped in favour of more modern approaches such as incremental development and eXtreme Programming (XP)? Many developers appear to show a preference for such modern approaches but there is not much published evidence to substantiate such choices. Software development and software engineering books discuss the alternatives but offer little in the way of direct comparison and explicit metrics that address the impacts on the product, project and people. Indeed metrics enable a deeper insight into the process by providing an empirical mechanism for objective evaluation.

The classic Waterfall model was refined in the early 1970s to cope with the larger and more demanding software projects characterised by a growing level of complexity. It was heavily influenced by early attempts to introduce structure into the programming process [1-7] and therefore offers a systematic development process leading to the orderly definition of artefacts. Many adaptations and adjustments have been made to the model leading to a variety of representations. Recent evidence suggests that it is still used extensively in many software development projects [8-13]. However, the proliferation of alternative models now offers a wide choice of perspectives and philosophies for development which are also being utilised by many practitioners.

Chief among them are incremental methods [14, 15], evolutionary methods [16], and more recently, XP and other agile methods [17-19]. XP is often viewed as a lightweight methodology focused on the delivery of business value in small fully integrated releases that pass all the tests defined by the clients. Reduced focus on documentation and control allows development teams to produce more creative solutions more rapidly [ibid.].

Given the reported benefits of XP in terms of working in small teams which appear to be more creative and result in higher user satisfaction levels [18, 20], it is not surprising that many managers and developers have adopted such practices. To date however, there is not much information on the direct comparisons between the different approaches in terms of the quality, functionality and scope of the resulting products. Nor is there much information regarding the impact on the project, the intermediary products and the development team.

The aim of this work was to investigate the impacts of different approaches through the evaluation of the metrics generated through their use. Fifteen teams of student-developers working on similar applications utilised four development approaches ranging from a version of the waterfall model, via incremental and evolutionary development to Extreme Programming. The resulting data on the relative merits, attributes and features of the products, the time spent in each stage (or type of activity) and the measures of requirements, design and programming outputs provide a start towards understanding the full impact of selecting a programming and management approach on the product, project, process and people and towards making better informed decisions about which one to apply under different circumstances.

2. Background

The skill set focusing on the life cycle of software engineering projects is critical to both understanding and practising sound development and management. Indeed life cycles are prominently featured within the Bodies of Knowledge of many different disciplines (including the PMBoK, SWEBOK and APM BOK) [21-23]. The technical life cycle identifies the activities and events required to provide an effective technical solution in the most cost-efficient manner.

The life cycle represents a path from the origin to completion of a venture. Division into phases enables managers to control and direct the activities in a disciplined, orderly and methodical way that is responsive to changes, adaptations and complications. Phases group together directly related sequences and types of activities to facilitate visibility and control thus enabling the completion of the venture. The project life cycle thus acts as a framework focusing on the allocation of resources, the integration of activities, the support of timely decision making, the reduction of risk and the provision of control mechanisms.

The benefits of using a life cycle approach identified by [24] include:

- attaining visibility,
- breaking work into manageable chunks,
- identifying the tasks,
- providing a framework for co-ordinating and managing,

- controlling project finance,
- identifying and obtaining correct resource profiles,
- encouraging systematic evaluation of options and opportunities,
- addressing the need for stakeholder review,
- providing a problem solving perspective,
- integrating activities,
- encouraging continuous monitoring,
- managing uncertainty, and
- providing a common and shared vocabulary.

Typical life cycle approaches to select from include sequential, incremental, evolutionary and agile approaches. Each is likely to be better suited to a particular scenario and environment and to result in certain impacts on the overall effort and the developed products. The general approaches as discussed in [24] are highlighted below in general terms.

Sequential Approaches. Sequential approaches (e.g. waterfall model, V-model) refer to the completion of the work within one monolithic cycle. [10]. Projects are sequenced into a set of steps that are completed serially and typically span from determination of user needs to validation that the given solution satisfies the user. Progress is carried out in linear fashion enabling the passing of control and information to the next phase when pre-defined milestones are reached and accomplished. This approach is highly structured, provides an idealised format for the contract and allows maximum control over the process. On the other hand, it is also resistant to change and the need for corrections and re-work. Note that some variations, as well as Royce's original formulation of the model [5], allow for revision and re-tracing and may also incorporate prototyping or other requirements gathering sequences encompassed within the overall sequence frame.

Sequential development is also referred to as serial engineering. The serial focus ensures interaction between phases as products are fed into the next step and frozen upon completion of a milestone. This essentially represents a comparison between the input to and the output of each phase. Sequential engineering also implies a long development sequence as all planning is oriented towards a single hand-over date. Explicit linearity offers a structured approach rich in order, control, and accountability. In order to overcome the impact of a late hand-over and delayed feedback, specific decision mechanisms, review point and control gates are introduced to ensure early discovery and correction of errors and a reduced aggregate cost to fix.

Incremental Approaches. Incremental approaches emphasise phased development by offering a series of linked mini-projects (referred to as increments, releases or versions) [15, 18, 25-28] working from a pre-defined requirements specification up front. Work on different parts and phases, is allowed to overlap throughout the use of multiple mini-cycles running in parallel. Each mini-cycle adds additional functionality and capability. The approach is underpinned by the assumption that it is possible to isolate meaningful subsets that can be developed, tested and implemented independently. Delivery of increments is staggered as calendar time progresses. The first increment often acts as the core product providing the functionality to address the basic requirements. The staggered release philosophy allows for learning and feedback which can modify some of the customer requirements in subsequent

versions. Incremental approaches are particularly useful when the full complement of personnel required to complete the project is not available and when there is an inability to fully specify the required product or to fully formulate the set of expectations.

Evolutionary Approaches. Evolutionary approaches [16, 29] recognise the great degree of uncertainty embedded in certain projects and allow developers and managers to execute partial versions of the project while learning and acquiring additional information and gradually evolving the conceptual design. Evolutionary projects are defined in a limited sense allowing a limited amount of work to take place before making subsequent major decisions. Projects can start with a macro estimate and general directions allowing for the fine details to be filled-in in evolutionary fashion. The initial implementation benefits from exposure to user comments leading to a series of iterations. Finite goals are thus allowed to evolve based on the discovery of user needs and changes in expectations along the development route. Projects in this category are likely to be characterised by a high degree of technological risk and lack of understanding of full implications by both stakeholders and developers. Evolutionary approaches are particularly effective in change-intensive environments or where resistance to change is likely to be strong.

Agile Approaches. Agile development is claimed to be a creative and responsive effort to address users' needs focused on the requirement to deliver relevant working business applications quicker and cheaper [17-19]. The application is typically delivered in incremental (or evolutionary or iterative) fashion. The agile development approaches are typically concerned with maintaining user involvement [30] through the application of design teams and special workshops. The delivered increments tend to be small and limited to short delivery periods to ensure rapid completion. The management strategy utilised relies on the imposition of *timeboxing*, the strict delivery to target which dictates the scoping, the selection of functionality to be delivered and the adjustments to meet the deadlines [31]. Agile development is particularly useful in environments that change steadily and impose demands of early (partial) solutions. Agile approaches support the notion of concurrent development and delivery within an overall planned context.

Each of the approaches described above appears to have clear benefits, at least from a theoretical perspective. However the variety of different approaches leads to a dilemma when it comes to selecting the most suitable one for a project. At the beginning of every project the manager is expected to commit to a development approach. This is often driven by past experience or other projects that are, or have been, undertaken by the organisation. Project managers are expected to select the most suitable approach that will maximise the chances of successfully delivering a product that will address the client's needs and prove to be both useful and usable. The choice should clearly relate to the relative merits of each approach. In practice, little work has been conducted in this area and aside from theoretical papers that compare and contrast some of the models listed above (see for example [32]), it is unusual to find studies comparing empirical result (e.g. specification vs. prototyping [33, 34])

In a recent text, Boehm and Turner define a general framework for selecting a development approach. Five attributes, (i.e. personnel, capabilities, requirements

volatility, group culture, group size, and application criticality), are used to indicate the degree to which a chosen development approach should be agile or plan-driven [20]. The text concludes that a small highly qualified group, working with volatile requirements, producing non-safety-critical software can successfully use agile methods while large mixed group working with stable requirements on large safety-critical system would be better off utilising a planned approach. The book however does not provide generic measures relating to each method beyond this basic guidance.

Recent case study results at IBM and Sabre Airlines focusing on the effects of adopting XP indicate a marked improvement in productivity (46%-70% in lines of code per person month) as well as significant decreases in pre- and post-release defect density. The team size at IBM was in the order of 7-11 and that used by Sabre Airlines, 6-10 members [35].

However, not enough is known about the impact of each approach on the development effort and the product quality. Moreover improvements in productivity such as the one reported above may not translate into enhanced user satisfaction levels or additional 'useful' functionality. More real data is needed in order to facilitate the comparisons between the approaches and the measurement of the results that emerge when each is adopted. Indeed, such measurements are likely to lead to better informed choices rather than the adoption of proposed 'fads and trends'. The rest of the paper describes the experiment, reviews the experimental results, frames them in terms of the discipline and generalises to draw the resulting conclusions.

3. The Experiment

The experiment was designed to investigate the impact of a development approach on the resulting product and its attributes. At the outset it was hypothesised that the four life cycle models under investigation (i.e. the V Model, Evolutionary Model, Incremental Model, and XP Model) would result in similar project outcome.

The experiment involved 55 student-developers working in fifteen teams developing comparable products from the same domain. The objective of the experiments was to investigate the differences in terms of development effectiveness and quality given the different approaches.

The experiment took place at University of Iceland during the winter 2003 – 2004 as part of a full year, two semester project. The project work was about 1/5 of the total workload of the students. All participants were Computer Science majors in the final year of their programme at the University of Iceland. Some of the students were part time students already working within the computer industry. Students had lectures twice a week and had one project meeting a week.

The products to be developed were interactive packages centred on a database system for home or family usage with a web-based user interface. The families of the students and indeed the students themselves were the "customers". The basic scope definitions of the projects were prepared in advance to ensure relative equality in terms of complexity and difficulty. However, all of the projects were open-ended and could only be partially "completed" in the allotted time. Each product was different. Users

(the student families and friends) were consulted through the process (on the basis dictated by their life cycle method), reviewed the system and came up with useful comments. Having fifteen distinct projects (as opposed to multiple versions of the same project) prevented the emergence of plagiarised solutions across the different groups.

A very brief description of the 15 projects is as follows: **Subscription SMS messenger service** – individual and groups can subscribe to receive SMS messages and reminders. **Entertainment** – information on films, videos/DVD, and games are entered and retrieved. subscribers rate items, actors etc. **Vehicles** – basic facts such as vehicle make, model, year and mileage are entered together with repairs, and operating costs. **Health** – system to keep track of health related information of each individual in a household such as illnesses, immunisations and treatments. **Exchange market** – users enter items that they want to exchange for others. Each user's credit or debit is tracked. **Diet**– members of a household enter information on what they eat and drink. Information on calorie intake, fibre, vitamins etc. computed. **Exercise** - users keep track of their fitness exercises. Records are kept over personal sport and exercise activities, what part of the body is being exercised etc. **CV** – keep track of vital information on members of a household and produce reports such as CV. **Properties** – keep track of properties owned by an individual such as real estate and cars. Record kept on operating and maintenance costs etc. **Security** – systemic safety and fire prevention and recording of access. **Digital photo album** – digital photos recorded, classified (in albums), displayed, and maintained. **Collections** – record and maintain collections of books, CD´s. Keep track of lending. **The home** – record information such as rooms, area, insulation, windows, heating consumption etc. **Home cooking** – recipe collection , nutrition values of ingredients and meals etc. **Accounts** – overview of checking- and savings accounts, stocks, bonds etc. of an individual. Loans and payments overviews and summaries.

All groups were asked to use JAVA and J2EE for transaction processing. Most used Sun public domain server tools for the deployment of J2EE and the Cloudscape database. A small number of the groups used JBoss server, MySQL database system and Borland development tools. Regardless of the platform selected, the groups were instructed to experiment their way through the technology with limited formal help. The result of the effort was a working product that can be viewed as working usability or technology prototype. (Note that the technology has matured considerably since the first run of the experiment. This year the students in the project course are using J2ME, Eclipse etc to develop applications for the mobile technology.)

No preferences regarding the choice of project, the team or the overall approach were accommodated. This is largely representative of real-life project environments. The 55 developers were randomly allocated to groups, each consisting of three or four developers. Each project title (with its general definition) was also randomly allocated to a group. Finally, in order to remove any bias, each group was also allocated a random development life cycle (from a set of four).

The set of software development methods included one representative from each class of approaches (Sequential, Incremental, Evolutionary, and Agile). The V-model (VM) (see for example [36] or IEEE830) was used as an example of a Waterfall-type sequential approach. The Incremental Model (IM) and the Evolutionary Model (EM) were explained in advance and used in accordance with the description in [36]. The agile approaches were represented by Extreme Programming (XP) as defined by Beck [19]. Each method was designated an expert leader (method champion). Students learned all methods and subsequently engaged in additional tutorials with their expert leader who was on hand to help with the use of the method and was also available to participate and consult in some of their scheduled meetings. As it turned out the IM groups did not write the overall requirements specification up front but rather wrote it incrementally. So there was not much difference in how the IM and EM groups worked.

Data was collected regularly by participants to ensure all stages of the different processes were covered. Students were encouraged to keep logbook recording all measures and faults. Measures were checked and confirmed by the relevant method champion to ensure accuracy and correctness. Students understood that the grades would not be linked to the time and productivity measurements recorded in their logs and therefore had no reason to misrepresent activities or events. They were further encouraged to record data and events as they happened rather than resort to memory.

The metrics collected by the teams included direct measures related to the life cycle (e.g. effort), the product (e.g. lines of code) and the intermediate artefacts (e.g. number of diagrams). In this way it was possible to focus on inputs (primarily in terms of human effort resources as cost was not a direct issue) and on outputs related to both the final product and the intermediate deliverables. In addition, the teams collected indirect measures (e.g. functionality, reliability and maintainability assessments). Following the experiment it was possible to derive a further class of measures related to both the product (e.g. Lines of Code per class) and the life cycle (primarily in terms of productivity rates) which further illuminate the relative performance of the different models.

It is worth noting that the V-model teams spent a lot of time on the requirements specification and the design activities and consequently did not have enough time left to implement everything that they had designed. The V-model requires sequential progression so that technology is only addressed during the implementation stage. The V-model groups started programming late only to discover that the J2EE server technology caused problems that forced the teams to spend significant periods in adjusting and re-adjusting to the technology thus affecting their overall time allocation and ability to deliver.

Intermediate results of this experiment were reported in earlier conference papers [37, 38]. They were primarily focused on the averages and the differences between averages across methods. This paper analyses the results following detailed statistical tests and places them in context based on the significance derived from the tests. It also adds derived metrics and frames the results in terms of effort distributions and method clustering as well as offering additional insights, reflections and limitations.

4. Results

This section describes and analyses the direct measurements from the different teams as well as some of the derived metrics related to the four development approaches.

The numerical results are presented in Tables 1 – 7 in the form of group averages. The underlying values, i.e. the individual values for the groups, are displayed in Appendix A in Tables 1A – 7A and referred to within the main text as needed - whilst the main discussion centres on the averages.

As part of the statistical analysis to establish the significance of the results, all of the project metrics were analysed using the R Statistical Package[1].

In order to reason about the significance of the differences in the group averages a null hypothesis is postulated:

> **H_0 -The averages for a particular attribute for the four models are alike when compared pair wise.**

In this work the H_0 is rejected "with 90% confidence" if the "p statistic" assumes a value less than 0.1 ($p < 0.1$) in pair wise comparisons using t tests with pooled standard deviation. In tables 1 – 7 (as well as in Tables 1A - 7A in Appendix A) statistically significant differences in pair wise comparisons for averages are shown in bold. It turns out that all of the significant differences relate to the XP model. It is to be note that the sample sizes *n* (i.e. the groups representing each method) are small, $n = 3$ or $n = 4$, so that the differences in the averages must be quite marked in order to be significant. Discussions in the paper are focussed on differences that are statistically significant.

One of the Incremental groups spent 8.4 Project Months (PM) (1275 hours) on the project. This group is taken to be a statistical outlier as their result is out of line with other groups and the overall time availability and is therefore dropped from the tables below. (Rationale: The value of 8.4 PM is well above the likely maximum of 6.6 PM as computed by the rule of thumb: *median + 1.5 Interquartile_Range*.) The overall average of the effort spent for the remaining groups is then 3.9 PM.

Boxplots[2] (box and whisker diagrams) are used where needed to provide a visual representation of basic statistics and their significance.

A. Time spent on the project

Table 1 shows the average effort data by development approach. (See also Table 1A for individual team performance.) Time spent is measured in hours and is divided into the seven general activities (rather than distinct phases) of requirements specification, design, coding, testing, review, repair (code correction and refactoring) and other (experimenting with technology etc), giving an overall total for each project. The total is converted into Project Months (PM) at 152 hours per PM. The groups are ordered

[1] R Statistical Package, http://www.r-project.org/ accessed on 14.05.05 and 18.02.2006.

[2] A boxplot is used to give a graphical overview over a group of values. It shows the minimum value as a segment at the bottom and maximum value at the top. The "box" is bounded below by the first quartile value (Q1) and above by the third quartile value (Q3) and shows the median value in between.

by the development model used – V-model (VM), Evolutionary Model (EM), Incremental Model (IM), and Extreme Programming (XP). The overall average (OAve) is also presented.

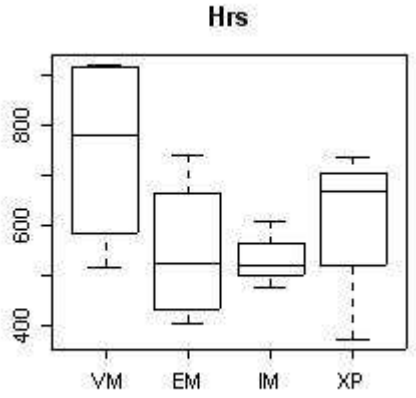
The only significant difference in the group averages (in bold) are found in the *requirements specification* activity where the XP groups spent considerably less time (***16.2***) as compared to that of the VM groups (**73.9**) and the EM groups (**67.8)** (i.e. the H_0 hypothesis of equal averages is rejected with 90% certainty.) Italics are used to signify the base of the comparison. In other words: XP < VM and XP < EM while VM ~ EM, VM ~ IM, EM ~ IM and IM ~ XP where the underscore signifies group average and the tilde "alike".

Table 1 Average group effort in hours by activity

Group	*ReqsSpec*	*Design*	*Code*	*Integr. & Test*	*Review*	*Repair*	*Other*	*Total hours*	*Total PM*
VM	**73.9**	68.6	206.1	35.6	56.9	60.4	246.5	748.0	4.92
EM	**67.8**	58.0	169.1	57.8	23.0	48.0	125.6	549.3	3.61
IM	43.8	51.2	185.7	40.7	37.7	53.8	121.6	534.5	3.52
XP	***16.2***	26.2	205.6	82.7	46.9	92.7	122.4	592.5	3.90
OAve	53.3	52.8	191.1	53.1	41.0	62.4	158.6	612.1	4.03

Figure 1 shows boxplots for the *total hours* of the 14 teams. While the boxplots for the methods appear to vary considerably, the differences in the overall averages are not significant. Clearly there is quite a large difference in the averages for individual group effort values. The longest time spent by a group, 919 hours, belongs to a V-model group whilst the shortest period, 373 hours, was exerted by an XP group (See Table 1A).

Figure 1 Distribution of total hours spent



The amount of work performed by each student varied considerably as is often the case in student projects. Furthermore few of the groups consisted of only three members while most had four. To counteract these differences it is seen appropriate to work with the percentile distribution of effort in order to reason about the effort patterns for the different methods.

Table 2 shows the group effort by activity as percentages. As noted for Table 1, the only significant difference in the averages is for the *requirements specification* effort.

The XP groups' values (***3%***) were significantly lower than that exerted by the VM groups (**10%**) and the EM groups (**12%**) as was the case in Table 1.

Table 2 Average group effort by activity as percentages

Group	*ReqsSpec %*	*Design %*	*Code %*	*Integr.& Test %*	*Review %*	*Repair %*	*Other %*	*Total %*
VM	**10**	10	27	5	7	8	33	100
EM	**12**	10	29	11	5	9	23	99
IM	8	9	35	7	7	10	23	99
XP	***3***	4	34	14	8	17	21	101
OAve	8.9	8.9	30.8	8.9	6.5	10.0	25.7	100

Other activities than *requirements specification* are statistically alike even if they show considerable fluctuation. It is noteworthy that the *coding* activity consumed 30.8% of the overall effort with the group averages closely aligned.

B. Requirements and high level design output

The output of the requirements specification and high level design (Tables 3 and 3A) was assessed in terms of the number of pages, Use Cases, screens, database diagrams, and DB tables. Note that Use Cases and XP "stories" were taken to be equivalent.

Table 3 Requirements and high level design outcome

Group	*Pages*	*Use cases*	*Screens*	*DB diagrams*	*DB Tables*
VM	**22.0**	7.5	**6.8**	1.0	3.3
EM	**19.0**	14.0	11.0	1.3	6.8
IM	**22.7**	14.7	**5.0**	1.3	4.3
XP	***5.7***	8.3	***17.7***	0.7	5.0
OAve	17.8	11.1	9.9	1.1	4.9

The XP groups produced an average of only ***5.7*** pages of requirements specification, which is significantly lower than that of any of the VM, EM or IM groups (at **22.0**, **19.0** and **22.7** respectively.) The XP groups did produce, on the other hand, a significantly higher number of user interaction screens (***17.7***) than the VM and IM groups (**6.8** and **5.0**).

C. Design Output

Design output (Tables 4 and 4A) is evaluated in terms of the number of design diagrams produced including overview diagrams, class diagrams, sequence diagrams, state diagrams, and other diagrams.

Table 4 Design outcome

Group	*Overview diagrams*	*Class diagrams*	*Sequence diagrams*	*State diagrams*	*Other diagrams*	*Total*
VM	1.0	2.5	2.3	5.3	1.8	**12.8**
EM	1.3	1.5	3.8	0.5	1.5	8.5
IM	2.3	3.7	**7.7**	0.0	2.0	**15.7**
XP	1.0	0.0	***0.0***	0.0	0.7	***1.7***
OAve	1.4	1.9	3.4	1.6	1.5	9.8

As the number of diagrams is relatively small it is easier to focus on the *total number of diagrams*. The number ranges between 1 and 22 as seen in table 4A in appendix A. The group producing 22 diagrams was utilising the V-model, while an XP group was responsible for the single diagram. The average for the Incremental groups is **15.7** diagrams, the V-model **12.8** and, followed by Evolutionary with 8.5 and XP with ***1.7***. XP teams produced significantly fewer diagrams during the design stage compared with the V-model and Incremental development.

D. Code size

The code size (Tables 5 and 5A) is measured in terms of Java classes and lines of code. Lines of code include lines produced in JAVA as well as JSP, XML and any other scripting code.

Table 5 Number of classes and lines of code

Group	*Java classes*	*Java*	*jsp*	*XML*	*Other*	*Total LOC*
VM	8.5	**1032**	1161	13	27	**2233**
EM	26.8	**1477**	1903	0	37	**3417**
IM	20.7	**1803**	922	36	14	**2776**
XP	27.7	***4836***	1662	987	254	***7740***
OAve	20.4	2140	1429	223	76	3867

The number of Java classes varies between 7 and 50 (see table 5A) with an overall average of 20.4.

In terms of *Java lines of code*, the XP model teams delivered significantly more code than the other models with an average of ***4836*** compared with **1803** LOC from the Incremental teams, **1477** from the Evolutionary and **1032** from the V-Model. The group averages for all other code (other than Java) are statistically alike.

The *total lines of code* produced by the teams are as follows: XP an average of ***7740*** LOC which is significantly more than that of Evolutionary with **3417**, Incremental **2776** and the V-model with **2233**. The comparison of the total lines of codes produced a striking result as the XP model teams delivered significantly more lines of code than any other team. The results would suggest that XP has a higher productivity rate in terms of the size of the delivered product. XP teams (as well as the other teams) produced additional code which was discarded during the duration of the project thus adding to their already impressive rate of the work.

E. Derived metrics

The derived metrics presented in Table 6 (and Table 6A) include Java lines of code per class, number of classes produced per Project Month, total lines of code per Project Month and page statistics. Page statistics include a) the requirements and design pages per month, b) pages of lines of code produced per month and c) the total pages produced per month i.e. the sum of a) and b). LOC pages per month figures were computed by dividing Total LOC by 50 (i.e. assuming an average of 50 code lines per page).

Table 6 Derived metrics

Group	*Java LOC / class*	*Classes/ PM*	*LOC / PM*	*RS&HD pages / PM*	*LOC pages / PM*	*Total pages / PM*
VM	122	1.7	**467**	**18.1**	**9.3**	27.5
EM	85	7.4	992	**21.6**	19.9	41.4
IM	88	5.9	820	**17.1**	16.4	33.5
XP	168	7.1	***2262***	***4.6***	***45.2***	49.9
OAve	114	5.5	1077	16.0	21.5	37.5

The number of *Java lines per Java class* are statistically alike. However they vary considerably between the different methods. Evolutionary method teams averaged 85 LOC per class. This was followed by the Incremental model with 88, the V-model with 122 and XP with 168 LOC per class. These differences are not statistically significant. The remaining derived metrics are all normalised by Project Months (PM).

The overall productivity measured in *LOC per PM* is 1077. XP leads with an average of ***2262*** LOC per PM, followed by Evolutionary methods with 992, Incremental with 820 and the V-model with **467**. Here the VM has significantly lower productivity than XP while other differences are not significant. XP clearly has the highest productivity with more than double the average rate. The Evolutionary and Incremental models are close to the average; however the V-model is clearly trailing in comparison with the average.

The average of the total number of *requirements and design pages* produced per PM places Evolutionary methods in the lead with **21.6** pages per month. This is followed by V-model with **18.1**, Incremental with **17.1** and XP with ***4.6***. XP is clearly in a different order producing around a quarter of the number of pages delivered by the other teams – XP produces significantly lower documentation output than any of the other.

The last column in table 6, the *total pages per project PM*, is noteworthy for the fact that here the statistical differences in the productivity have disappeared.

F. Quality metrics

Quality was assessed following ISO9126 product attributes and focusing on the five areas of Functionality, Reliability, Usability, Efficiency and Maintainability. Each attribute was assessed on a scale of 0-10 following the standard (with 10 being best). Each of the groups assessed the close-to-final outcome of two other groups that were using different development models. Not much time was available for this part of the experiment so that the results are not reliable. The measurement of Reliability and

Efficiency was not really significant because the resulting products were of prototype nature. Therefore these measurements are not reported (Table 7 and 7A).

Table 7 product quality assessment on scale 0-10

Group	*Functionality*	*Usability*	*Maintainability*
VM	7.9	7.4	8.9
EM	7.9	7.3	7.9
IM	8.3	8.3	7.3
XP	8.0	7.7	7.7
OAve	8.0	7.7	8.0

The group averages for each of the three "ilities" reported lie very close to each other making the difference statistically insignificant. The fact that the averages are much the same is a result in itself: The groups, when assessing the work product of the groups using different development model, did not see noticeable difference in the end product.

Maintainability was defined as ease of reading the documentation and code in order to find out how to make amendments to the system. The groups assessing the XP products initially indicated that they did not know where to start because of the lack of documentation. But when they started reading the code they realised that the systems were "quite easy to understand".

G. Defects

Each team was meant to collect defect data and classify it into 7 types of defects classified according to the Orthogonal Defect Classification (ODC) scheme [39, 40]. This data turned out to be heterogeneous and inconsistent and is therefore not reported.

5. Conclusions, observations and limitations

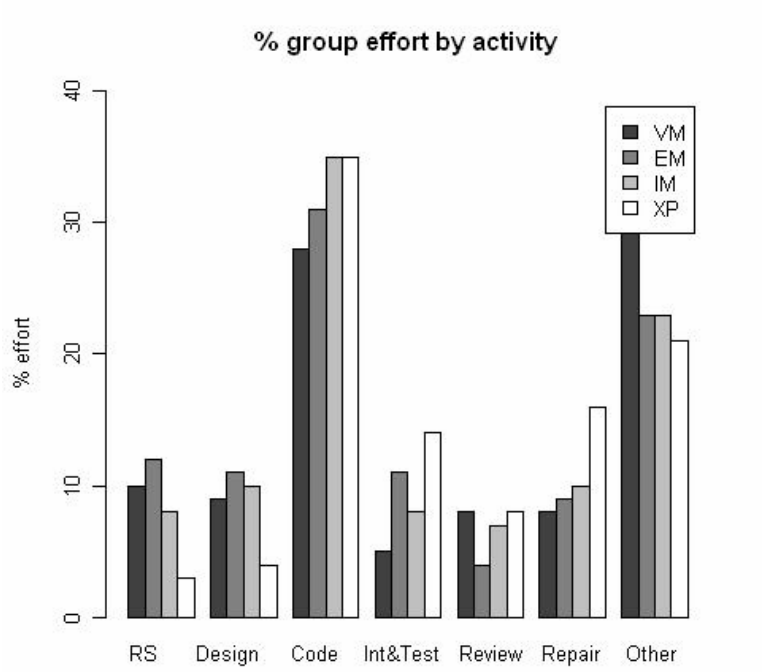
The results of this experiment provide some useful quantitative information on the relative merits of different development methods and on the impact they can have on different activities. The experiment conducted here is viewed as a first step towards providing a clearer understanding of the issues related to the choice of development approaches. The results can be sensitive to the performance of individuals within groups (of three or four students) and to the synergy created within any group, as there were typically only four groups of each life cycle approach (with the exception of XP). Indeed, one group included a number of accomplished programmers working for the Iceland Telephone company and a bank. Nonetheless, the results are significant in highlighting trends and providing a comparison between the different approaches and the trends that emerge from the study.

I. Time spent on project

Despite the fact that all projects addressed comparable problems, V-model projects took somewhat longer than the other projects. The other three models fell within 10% of each other. The requirements stage in the V-model consumed more time than the requirements activities in the other models. Similar trends were observed in design activities. In terms of the significance of the results, XP only consumed a minor proportion of the effort during the requirements (and design) activities compared to the other models.

Figure 2 depicts the distribution of the effort spent by the four categories of groups. It is interesting to note that XP teams took the lead in terms of hours spent in testing and code correction. The effort of the Waterfall teams dominated the requirements, design and even the programming activities. Integration and testing required relatively little effort from the Waterfall teams (and Incremental teams), presumably due to the level of detailed planning and additional time spent during the earlier stages of requirements and design, possibly also indicating earlier discovery of errors [10, 41].

Figure 2. Effort distribution by activity



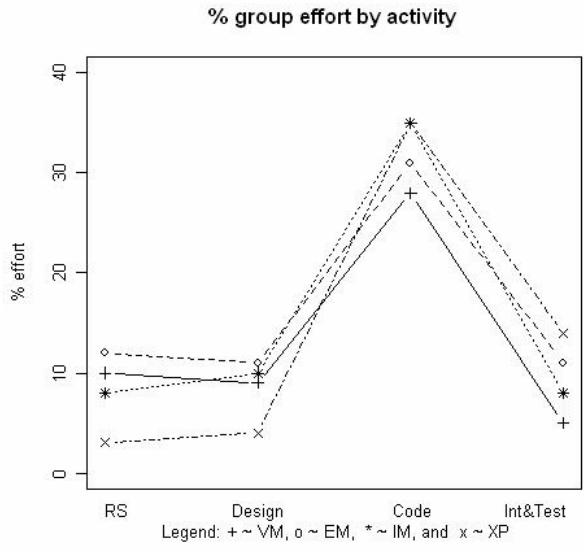
In terms of all projects, the activity of requirements was responsible for roughly 9% of the time spent (see Table 2). The activity of design also consumed approximately 9% of the overall time. In XP however, requirements resulted in 3% and design in 4%. Overall, programming used up 31% of the time, integration and testing 9%, reviewing 7%, repair 10% and other activities 26%. It is important to note that activities not directly considered as part of the life cycle (‘other’) still accounted for over a quarter of the time and must therefore be planned for in the early stages of the project. This is roughly in line with industrial rules of thumb for additional (and non-technical) activities. Note however that the time spent on ‘other’ activities by V-model teams was on average double the time spent by the other teams reflecting the additional effort and adjustments resulting from deferring the interaction with technology to the later stages.

Excluding the non-technical activities produces an overall profile for all teams of 12% for requirements, 12% for design (against an expectation of some 20%), 41% coding, 12% integration and testing, 9% review and 13.5% for repair. Indeed, one could argue that about a third of the technical effort was dedicated to quality activities including testing, integration, review and repair, which is roughly in line with the typical rule of thumb for quality activities. The early definition activities of requirements and design thus seem to add up to just under a quarter of the overall development effort. Overall, the figures are still slightly high in terms of coding (cf. [42] , 15%), and marginally low in terms of design (where, according to the literature, we could have expected a figure in the range of 15-20%).

Figure 3 shows the distribution of the effort for the activities of requirements specification, design, code and integration and test which are viewed as typical activities in a development context. The familiar shape of a Norden-Rayleigh curve (with significant coding humps) emerges here for each of the four model types [43]. It is worth noting the slight drop between the requirements and design efforts in both the incremental and the evolutionary methods due to some of the preparatory work being completed upfront.

It is also interesting to compare the relative positioning of the XP model compared to the other models. XP appears to generally conform to the effort distribution pattern expected from a Norden-Rayleigh curve. However, the curve starts at much lower point for the early activities (representing less expended effort during these activities) but then rises to the maximum level of effort during the coding activity thus representing the steepest climb rate out of the four models presented. This supports the reported focus of XP and other agile methods on delivering useful code and spending less upfront effort through the avoidance of exhaustive analysis and documentation [19, 30]. It is also interesting to note the higher level of integration and testing effort that goes into XP. The lighter effort upfront, combined with the heavier loading in terms of coding, integration and testing still appear to provide a general Norden- Rayleigh curve but with a much steeper diagonal tilt.

Figure 3. Effort distributions by development activity



The results seem to confirm the notion that XP requires less effort during the initial stages, in particular during the requirements activity. Repair activities in XP consumed more resources than in the incremental and evolutionary models. In closing it is also significant to note that 'other' activities were responsible for consuming significantly less hours in XP, than they did in the V-model.

It may be interesting to explore in subsequent work why the design effort is lower than expected. This may be related to the fact that these are student projects or reflect a general reluctance to tackle complex design issues (making an even stronger case for agile methods). In re-visiting Figure 3, one can also discern that the V-model and the evolutionary model show a slight deviation from the standard Norden-Rayleigh distribution by showing a decline between the requirements effort and the design effort level once again emphasising the below-expectation effort level for design.

II. Requirements and Design outputs

The V-model and the incremental model produced a significant amount of pages, words and lines of requirements. The Evolutionary method was not too far behind. XP produced less than a quarter of the number of pages, roughly a sixth of the number of words and between a seventh to an eighth of the lines of specification produced by the V-model and the Incremental model. In other words, XP results in significantly less pages of requirements and in less words being used to describe these requirements (not surprising considering the reduced input in terms of effort hours).

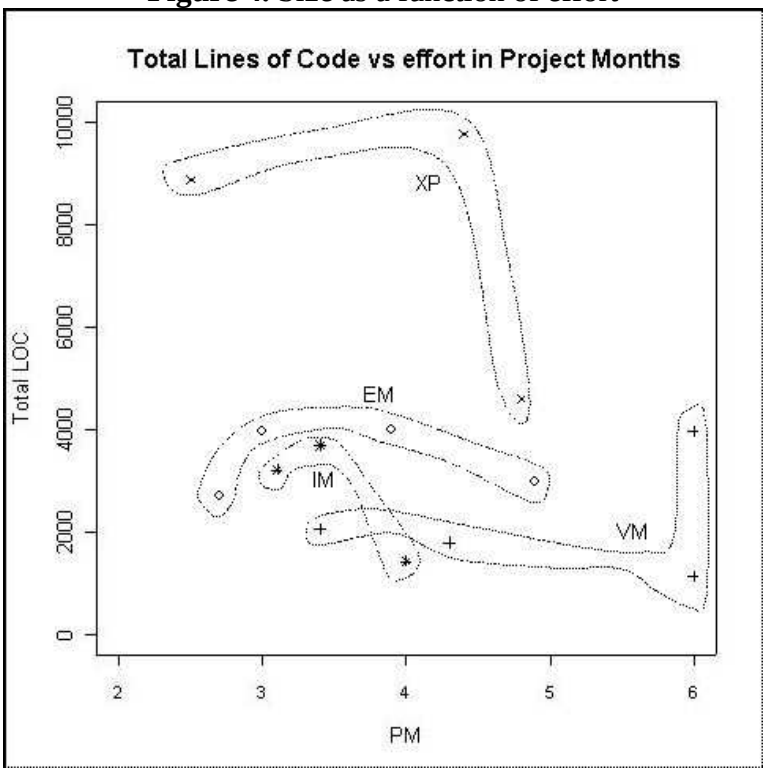
XP produced significantly more screens than the other methods. In terms of Use Cases both the V-model and XP teams produced significantly less Use Cases than the Incremental and Evolutionary teams. XP delivered on average less than two design diagrams compared with almost 16 produced by the Evolutionary and 13 by the V-model teams.

III. Lines of code

XP produced on average 3.5 times more lines of code than the V-type model, 2.7 times the lines of code produced by the Incremental model, and 2.2 times more code than the Evolutionary teams. This significant trend is also reflected in terms of the number of LOC produced in Java and in XML. Figure 4 offers a size and effort comparison between the different groups. It is easily discernible that the XP teams developed the greatest number of lines.

Boehm, Gray and Seewaldt [33] showed that with traditional development and with prototyping, the development effort was generally proportional to the size of the developed product. While this still appears to roughly apply to most groups, the change of paradigm enabled by XP seems to offer significantly greater productivity as measured by size in LOC from two of the XP teams. XP therefore in comparison with more traditional development approaches appears to defy this relationship.

Figure 4. Size as a function of effort



Another way of viewing the product size is through the classification of size categories originally provided by Boehm and by Fairley [41, 44]. Both Boehm and Fairley asserted that a small product is in the range of 1-2K LOC. While Boehm suggested that a typical intermediate/medium product is the range of 8-32K LOC, Fairley adjusted the range to 5-50K LOC. Moreover, Fairley asserted that medium-sized projects require teams of 2-5 working for 1 to 2 years. (Note that while the current typical sizes of programs are much larger, the classifications can still be used as a comparative measure – especially in our student project context.) The XP groups are the only teams to have produced products that would clearly qualify as medium-sized products according to both the Fairley and Boehm criteria. Significantly, these products were also viewed as the most comprehensive and the most satisfactory products as perceived by the users. Given the same amount of time, XP teams were thus able to build on the feedback from small and frequent releases, resulting in an intermediate/medium product (compared to the smaller products delivered by the other teams). The ability to involve users and to deliver on a more regular basis seem to have resulted in additional useful functionality and to have led to a greater level of acceptance and satisfaction from the users thereby making the resulting product more significant in terms of functionality, as well as larger overall. Layman's work [35] provides additional support to the assertion of increased customer satisfaction associated with XP practices.

IV. Productivity

XP teams were 4.8 times more productive in terms of LOC/PM than the V-model teams, 2.8 more than the Incremental teams and 2.3 more than the Evolutionary teams as can be computed from the data in Table 6. So the difference in productivity is more

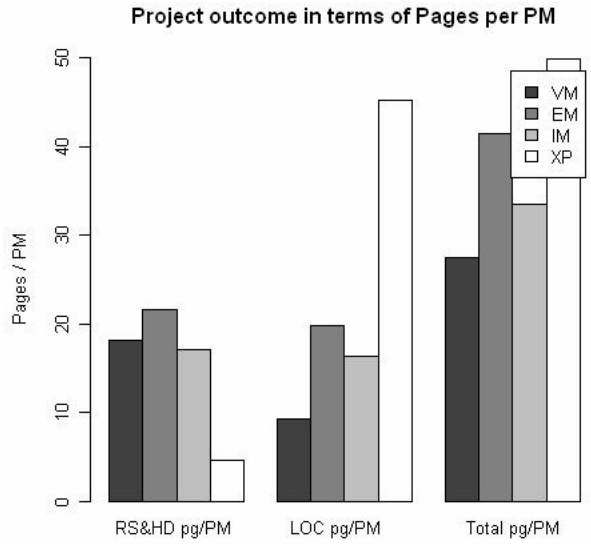
pronounced than that observed for product size. Layman [35] also reported significant differentials in productivity, albeit slightly less pronounced). Figure 4 shows the relative clustering of the products developed using different approaches and enables relative comparisons between the clusters. XP can be seen to come out on top, with the V-model clustered at the bottom. It would thus appear that incremental and evolutionary methods appear to outperform the V-model in terms of the relationship between the size of the product and the developed effort. However, the XP alternative produces a radically better ratio.

The picture is repeated for the derived function of number of pages of LOC. As noted earlier, XP teams produced significantly less pages of documentation related to the requirements and design activities. When adding the early activities in terms of pages to the coding pages, the gap between XP teams and the other teams is significantly reduced (i.e. spending less time on earlier activities enabled the XP teams to focus on delivering additional functionality). Note however, that LOC measures tend to ignore discarded code which is an inherent part of agile development thus suggesting an even higher productivity from the XP teams.

The average productivity rate is 1077 LOC/PM, with XP dominating with an average of 2262 LOC/PM. Industrial productivity is often taken to be in the range of 200 to 500 LOC/PM. With the exception of the V-model, all methods averaged above the upper range of the expected productivity level. However, XP clearly offered outstanding productivity. The average productivity is also computed at 5.5 classes/PM (with the V-model clearly trailing with 1.7 classes/PM). Kan [45] reports productivity in C++ and Smalltalk projects in the range 2.8 to 6 classes/PM. With the exception of the V-model teams, all models offered average performance at or above the top end of the productivity figure put forward by Kan. All V-model teams performed at or below the lower limit of Kan's range (in fact only one V-model team was within the range with the other three performing clearly below it).

The last three columns of Table 6 pertain to average productivity normalised with Project Month. This data is displayed in Figure 5.

Figure 5. Productivity in terms of Pages per PM



It was noted in the discussion of Table 6 that XP groups produced significantly less requirements and design documentation on one hand but significantly more code on the other hand. VM teams on the other hand spent a large proportion of their time during the earlier definition activities leaving themselves less time to work on subsequent phases. Table 6 looks at the quantity of pages of requirements and design that were produced by the different teams (in addition to the coding productivity). Once the total outputs of each group are taken into account, *the difference in the total number of pages delivered by the different groups is not significant* as the above graph indicates.

V. Quality

The quality metrics, which are not totally reliable, appear to show a preference for the V-model and the Incremental model but these results may not be sensitive enough or reliable enough without additional validation. Indeed, statistical analysis confirms that the results are too close to read much into them. Moreover, the subjective nature of the assessment renders the quality results no more than an interesting curiosity. However, it is interesting to note that the average reading for perceived product maintainability does not appear to penalise the XP products even if they only have the brief “stories”, unit tests, and then the code as their documentation.

VI. The experiment

The experiment yielded valuable results. The teams delivered working prototypes (not final quality products). The technology used may have played a role. The teams using Sun public domain server tools and Cloudscape were working with tools that were not fully mature products. A few teams utilised the JBoss server, MySQL and Borland tools which are more mature and seemed to work better.

The need to experiment with technology proved time-consuming for some of the groups. This became an issue for the V-type model teams as the experimentation was delayed until after the requirements were fully understood and specified and the design was stable. The impact of exploring the technology meant that these teams were forced to re-assess some of their design solutions in line with their emerging understanding of the technical environment. Indeed, this touches on the relationship (and intertwining) between design and implementation environment and the need to integrate the two [46]. The discussion of a problem is often enhanced by the consideration of design and implementation concerns. The constraints imposed on a solution by later stages need to be acknowledged and addressed to reduce the conflicts that will need to be resolved at a later stage. Methods that create functional barriers and that do not allow a relationship, albeit rudimentary to the notion of the solution may thus play a part in delaying essential interactions thereby arresting progress and requiring subsequent rework cycles to rectify the effects of the separation.

Clients were asked to assess the final delivered products in terms of comprehensiveness and satisfaction. The most comprehensive solutions, resulting in the highest level of satisfaction from the users (both in terms of the product and the development process), were provided by the teams using XP.

VII. Limitations

The experiment involved 15 teams working on comparable projects utilising four different models. As a result the sample size for each model is three to four groups (in line with other known experiments in this area using two, three or four groups in each category, [33, 34]). Despite the limited number of data points the experiment offers a quantitative investigation of the extent to which the development approach affects the outcome of a project and an experimental comparison of the different software development life cycles and methods. The empirical findings are therefore viewed as a systematic interpretation offered as part of a more comprehensive study, rather than as conclusive answers.

The choice of a life cycle is related to the type of problem being solved. Given that the experiment attempted to explore the impact of the life cycle on the development effort in 15 parallel projects, there would inevitably be a certain degree of a limiting effect in the experimental design in trying to keep all teams working on similar problems to facilitate comparison. This criticism could equally apply to other similar experiments [33, 34] and comparative work [35]. Indeed this is a fundamental flaw of a lot of empirical work that attempts to derive comparisons between methods and approaches in a real-life environment. Nonetheless, it is useful to have a direct comparative measure between the methods. In the experiment, XP teams were able to apply additional cycles and to further clarify the context and build in additional features to further satisfy their users, whilst the V-model teams in contrast, delivered products that come closer to the original definition of what was originally required as they spent longer during the conceptual definition phases.

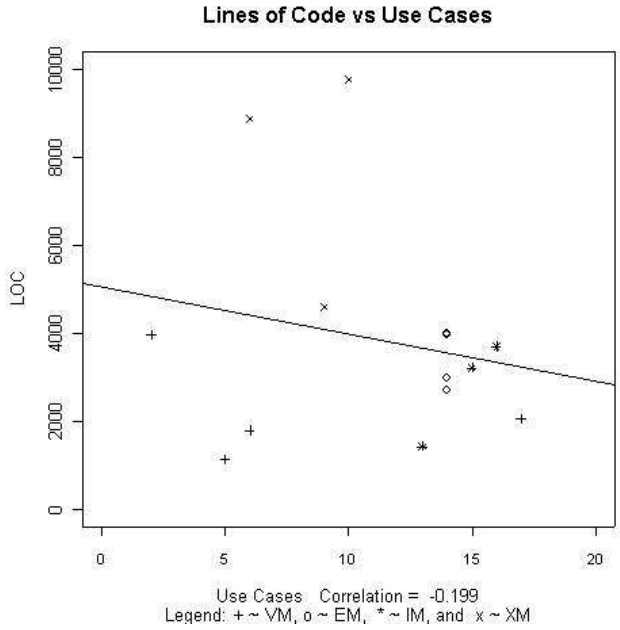
Employing students as developers offers many benefits to all parties. Most significantly, it enables students to interact with real development tasks, to gain experience in teamwork and to work on non-trivial problems from beginning to end thus embracing the entire life cycle. From the point of view of the experiment it enables direct experimentation in the development of similar or comparable products through the use of alternative methods and approaches. However, this also implies that the experiment was conducted in educational settings, rather than an industrial environment. Consequently, certain aspects of the environment which are normally associated with XP practices, such as sustainable pace, on-site customer, continuous communication and open workspace may have been slightly compromised. On the other hand the definition of the work and its relevant context meant that participants were stakeholders themselves. It is impossible to surmise whether the XP results would have been even more pronounced under the full conditions recommended as best practice for XP development.

VIII. Software Metrics

Common metrics focus on LOC's or function points. The experiment was primarily based on LOC or size measures. However, it also attempted to identify other measures such as use-cases. Use-cases can provide a normalisation measure similar to LOC or function points. In common with function points (and unlike LOC) they are defined early within the process and are also independent from the programming language thus providing an indirect measure of the features and functions within the system. They therefore have the potential to provide a method of direct estimation that can be utilised during the early stages. Whilst there is no standard size for use-cases, the number of use-cases is viewed as directly proportional to the size of the application in LOC and the number of test cases that will have to be designed [47].

To show this direct correlation between the number of use-cases and the overall size, the experiment recorded the use-cases generated by each group. The number of use-cases in the incremental and evolutionary teams is very similar. V-model and XP however, produced significantly less use-cases (at about half the ratio of the other methods). The results do not seem to support the common assertion regarding a direct relationship between use-cases and LOC size. The incremental teams with the highest average for use-cases produced almost a third of the code delivered by some of the other teams with a lower number of use-cases. XP teams on the other hand, had a much lower number of use-cases, but significantly more code than other teams (see Figure 6). The experiment does not support the position of use-cases as an accurate predictor. This suggests that much more work needs to be done before use-cases can be relied upon as a reliable normalisation measure.

Figure 8. Use case to size correlation



More generally, direct measures related to the life cycle, the product and the artefact (i.e. effort, LOC and diagrams respectively) were relatively easy to record and were viewed as reasonably accurate. As a result managers can have adequate figures relating to the input into the process and to the delivered outputs. Indirect measures proved a greater challenge and the experiment will require improved mechanisms for measuring them in the future. Indeed, it would appear that developing reliable quality

(and reliability) measures is strongly dependent on the methods and procedures for capturing the data as well as on the participants. The derived measures, such as the ones calculated following the conclusion of the experiment, provide greater visibility into issues related to productivity. The development of reliable indirect measures will further enable the detailed assessment of quality and its attributes in relation to other product and process parameters.

The majority of software development efforts fail to use measurements effectively and systematically. Measurement typically facilitates improvement and control and is used as the basis for estimates, as a method for tracking status and progress, as validation for benchmarking and best practice, and as a driver in process improvement efforts. In order to improve, software engineers require on-going measures that compare achievement and results. However, the type of measures utilised in this paper is useful in addressing and making decisions at a higher level by providing insights into different life cycles and their impacts. It can therefore be viewed as meta-measurement that plays a part in comparing the attributes, impacts and merits of competing approaches to development. This type of comparative work is also very rare in a practical software engineering context. Shining the torch where light does not normally go can open a new avenue for exploring the impact of major decisions about which life cycle to adopt and lay the foundation for understanding the impact and suitability of such decisions.

IX. Concluding comments

Agile methods, including XP, are predicated on the assumption of smaller teams working more closely, utilising instantaneous feedback and fostering knowledge sharing. The focus on smaller iterations and limited documentation makes them ideal as learning-focused and client- or business satisfaction- centred approaches. Intriguingly, the XP teams, the only teams that were meant to work ‘in the small’, have delivered the biggest and most comprehensive products. The V-model in contrast, appears to have necessitated the most time, to have used the most words and correspondingly to have provided the fewest diagrams. It also appears to have resulted in a delayed consideration of technology, further delaying the progress of the teams.

XP incorporates a few rules and a small number of best practices which are relatively simple to apply. It is particularly suitable for small to medium sized systems development, particularly in change-intensive or partially understood environments where delivery of business value is viewed as crucial (and the production of detailed documentation is not a prime concern). Unlike the alternative approaches considered in this study, XP is neither process-centred nor prescriptive. Methodologies like XP, offer the flexibility to be shaped according to the context and project needs to support the delivery of relevant value.

Selecting the most suitable method is contingent on the context and participants. The direct comparison between the four approaches offers an interesting way of quantifying the relative impacts of the various approaches on the product. Whilst incremental and evolutionary approaches have been offered as improvements to sequential approaches, the added comparison with XP is instructive. XP is currently offered as a lightweight alternative to the sequential notion. The direct comparison

between the four approaches therefore provides a quantitative basis for beginning to consider the impact of the different approaches thus bridging the gap between the descriptions of the different life cycles and the need to make an educated choice based on the likely impact of the approach. The comparison yields interesting results and comparative measures (e.g. V-model teams and XP teams produced significantly less Use Cases than incremental or Evolutionary teams).

In the experiment XP teams produced solutions that contained additional functionality. Indeed, in terms of significant results, their products can be characterised as consisting of the largest number of screens and the most lines of code. In terms of the process, they can be said to have spent the least time on requirements and consequently to have produced the least number of pages of requirements. They also generated significantly less diagrams. Early work on how programmers spend their time seemed to indicate that a very small proportion of a programmer's time (13-15%) is spent on programming tasks [44]. Methodologies like XP attempt to overcome that by optimising the time available for programming (e.g. minimal documentation) resulting in enhanced output (more code and more screens) that is delivered more rapidly.

X. Experience

The student experience was evaluated by way of a survey conducted at the end of the final semester that looked at the results. Most students working in Incremental, Evolutionary and XP teams were content with their model. Intriguingly, ALL students in groups utilizing the V-model indicated a strong preference towards using a different model to the one they were allocated and were therefore less satisfied with the process (thus matching the perceptions of their clients). Layman [35] also reported higher levels of developer morale associated with XP teams compared with other models.

XI. Future Work

It is intended to continue with the experiments and the group projects. The lessons learned from the first year of the experiment will result in a number of small changes which will be implemented in the next round (currently under way):

- Following the strong preference against the use of a sequential model the groups will be allowed to choose their development model. It will be interesting to see how many of the teams will select some form of agile methods.
- The suite of tools selected will be based on mature and more unified technology. Groups will be instructed to use Eclipse platform-based Java and J2EE with integrated use of Ant, CVS, JUnit, and MySQL based on the JBoss server. An OO metrics plug-in for Eclipse will also be utilised.
- To ensure that all teams deliver useful products there will be firm dates for delivering partial increments (two in the first semester, three in the second).
- Participants will be encouraged to record additional measurements to enable direct comparisons between different sets of measures and to assess the feasibility and impact of different systems of measurement focusing on improved recording of LOC, function points, use-cases, and quality attributes.

Some additional areas of investigation include adding criteria to account for refactoring in terms of effort (thereby testing the assertion of 25% of effort in XP is dedicated to refactoring) and in terms of abandoned code discarded during refactoring and continuous design improvement.

It is clear that the large number of variables makes a simple decision about the 'ideal method' and the appropriate set of tools difficult, if not impossible. The different metrics reveal that each method has relative strengths and advantages that can be harnessed in specific situations. Experiments such as this make a significant contribution to understanding the relative merits and their complex relationships. As the experiment is improved, and hopefully repeated elsewhere, a clearer picture of the issues, merits and disadvantages is likely to emerge and a deeper understanding of the role and application of each life cycle method will hopefully ensue.

References

1. Canning, R.G., *Electronic Data Processing for Business and Industry*. 1956, New York: John Wiley.
2. Canning, R.G., *Installing Electronic Data Processing Systems*. 1957, New York: John Wiley.
3. Bennington, H.D., *Production of Large Computer Programs.* Annals of the History of Computing, 1956. **5 Oct. 1983**(4): p. 350-361.
4. Hosier, W.A., *Pitfalls and Safeguards in Real-Time Digital systems with Emphasis on Programming.* IRE Transactions on Engineering Management, 1961: p. 91-115.
5. Royce, W.W. *Managing the Development of Large Software Systems: Concepts and Techniques*. in *Proceedings, IEEE WESCON, August 1970*. 1970.
6. Laden, H.N. and T.R. Gildersleeve, *System Design for Computer Applications*. 1963, New York: John Wiley.
7. Farr, L.A., *Description of the Computer Program Implementation Process*. 1963, SDC Technical Report.
8. Neill, C.J. and P.A. Laplante, *Requirements Engineering: The State of the Practice.* IEEE Software, 2003. **20**(6): p. 40-45.
9. Laplante, P.A. and C.J. Neill, *The Demise of the Waterfall Model is Imminent and other Urban Myths.* ACM Queue, 2004. **1**(10): p. 10-15.
10. Pressman, R.S. and D. Ince, *Software Engineering: A Practitioner's Approach*. 5 ed. 2000, Maidenhead: McGraw-Hill.
11. Chatters, B.W. *Software Engineering: What do we know?* in *FEAST 2000, July 2000*. 2000. Imperial College, London.
12. Dalcher, D., *Towards Continuous Development*, in *Information Systems Development, Advances in Methodologies, Components and Management*, M. Kirikova and e. al., Editors. 2002, Kluwer: New York. p. 53-68.
13. Cusumano, M.A., et al., *A Global Survey of Software Development Practices*. 2003, MIT: Cambridge, Ma. p. 1-17.
14. Mills, H.D., *Incremental Software Development.* IBM Systems Journal, 1980. **19**(4): p. 415-420.
15. Graham, D.R., *Incremental Development and Delivery for Large Software Systems.* IEEE Computer, 1992: p. 1-9.
16. Gilb, T., *Evolutionary Development.* ACM SIGSOFT Software Engineering Notes, 1981. **6**(2): p. 17.
17. Cockburn, A., *Agile Software Development*. 2002, Boston, MA: Addison-Wesley.
18. Laraman, C., *Agile and Iterative Development: A Manager's Guide*. 2004, Boston, MA: Addison-Wesley.

19. Beck, K., *Extreme Programming Explained: Embrace Change*. 2000, Boston, MA: Addison-Wesley.
20. Boehm, B.W. and R. Turner, *Balancing Agility and Discipline - A Guide for the Perplexed*. 2004: Addison-Wesley.
21. PMI, *A Guide to the Project Management Body of Knowledge*. 2000 ed. 2000, Newton Square, PA.: Project Management Institute.
22. Dixon, M., *APM Project Management Body of Knowledge*. 4th ed. 2000, High Wycombe: Association for Project Management. 68.
23. Bourque, P. and R. Dupuis, *A Guide to the Software Engineering Body of Knowledge SWEBOK*. 2001, Los Alamitos, CA: IEEE Computer Society.
24. Dalcher, D., *Life Cycle Design and Management*, in *Project Management Pathways: A Practitioner's Guide*, M. Stevens, Editor. 2002, APM Press: High Wycombe.
25. Mills, H.D., *Top-Down Programming in Large Systems*, in *Debugging techniques in Large Systems*, R. Ruskin, Editor. 1971, Prentice-Hall: Englewood Cliffs, New Jersey. p. 41-55.
26. Laraman, C. and V.R. Basili, *Iterative and Incremental Development: A Brief History*. IEEE Computer, 2003. **36**(6): p. 47-56.
27. Denne, M. and Cleland-Huang, *Software by Numbers:-- Low Risk, High-Return Development*. 2004: Prentice-Hall.
28. Benediktsson, O. and D. Dalcher, *Effort estimation in incremental software development.* IEE Proceedings Software, 2003. **150**(6): p. 251-358.
29. Gilb, T., *Principles of Software Engineering Management*. 1988, Wokingham: Addison Wesley.
30. Agile_Alliance, *Agile Manifesto*. 2001, The Agile Alliance.
31. Stapleton, J., *DSDM Dynamic Systems Development Method*. 1997: Addison-Wesley.
32. Davis, A.M., E.H. Bersoff, and E.R. Comer, *A Strategy for Comparing Alternative Software Development Life Cycle Models.* IEEE Transactions on Software Engineering, 1988. **14**(10): p. 1453-1461.
33. Boehm, B.W., T.E. Gray, and T. Seewaldt, *Prototyping vs. Specifying: A Multiproject Experiment.* IEEE Transactions on Software Engineering, 1984. **SE-10**(3): p. 290-303.
34. Mathiassen, L., T. Seewaldt, and J. Stage, *Prototyping vs. Specifying: Principles and Practices of a Mixed Approach.* Scandinavian Journal of Information Systems, 1995. **7**(1): p. 55-72.
35. Layman, L. *Empirical Investigation of the impact of Extreme Programming Practices on Software Projects*. in *OOPSLA '04*. 2004. Vancouver, British Columbia: ACM Press.
36. Pfleeger, S.L., *Software Engineering: Theory and Practice*. 2 ed. 2001, Upper Saddle River, New Jersey: Prentice-Hall.
37. Benediktsson, O., D. Dalcher, and H. Thorbergsson. *Working with Alternative Development Life Cycles: A Multiproject Experiment.* in *Thirteenth International Conference on Information Systems Development - ISD'2004, Advances in Theory, Practice and Education*. 2004. Vilnius, Lithuania.
38. Dalcher, D., O. Benediktsson, and H. Thorbergsson. *Development Life Cycle Management: A Multiproject Experiment*. in *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*. 2005: IEEE Computer Society Press.
39. Chillarege, R., et al., *Orthogonal Defect Classification – A Concept for In-Process Measurements.* IEEE Transactions on software Engineering, 1992. **18**(11): p. 943-956.
40. Butcher, M., H. Munro, and T. Kratschmer, *Improving Software Testing via ODC: Three Case Studies.* IBM Systems Journal, 2002. **41**(1): p. 31-44.
41. Boehm, B.W., *Software Engineering Economics*. 1981, Englewood Cliffs: Prentice Hall.

42. Macro, A. and J.N. Buxton, *The Craft of Software Engineering*. 1987, Wokingham: Addison Wesley.
43. Putnam, L.H., *A General Empirical Solution to the Macro Software Sizing and Estimating Problem.* IEEE Transactions on Software Engineering, 1978. **SE-4**(4): p. 345-361.
44. Fairley, R., *Software Engineering Concepts*. 1985, New York: McGraw-Hill.
45. Kan, S.H., *Metrics and Models in Software Quality Engineering*. 2003, Boston: Addison-Wesley.
46. Swartout, W. and R. Balzer, *On the Inevitable Intertwining of Specification and Implementation.* Communications of the ACM, 1982: p. 438-440.
47. Pressman, J.L., *Software Engineering: A Practitioner's Approach*. 6 ed. 2004, New York: McGraw-Hill. 912.

Appendix A: detailed results

Appendix A provides the detailed tables showing the results obtained by each group.

Table 1A shows the effort data in hours as reported by the groups divided into life cycle approach and the specific phase showing the stage components and overall total by team.

Table 1A Individual group effort in hours by activity

Group	*ReqsSpec*	*Design*	*Code*	*Integr. & Test*	*Review*	*Repair*	*Other*	*Total hours*	*Total PM*
VM1	65	76	111	11	23	36	194	515	3.4
VM2	68	54	291	13	75	108	301	907	6.0
VM3	97	80	231	31	97	68	317	919	6.0
VM4	66	66	192	89	34	31	175	651	4.3
Ave	**73.9**	68.6	206.1	35.6	56.9	60.4	246.5	748.0	4.92
EM1	84	93	255	63	6	75	165	741	4.9
EM2	37	73	116	39	14	63	122	463	3.0
EM3	83	46	230	48	34	35	112	588	3.9
EM4	68	20	76	81	38	19	104	406	2.7
Ave	**67.8**	58.0	169.1	57.8	23.0	48.0	125.6	549.3	3.61
IM1	57	52	259	30	14	52	56	520	3.4
IM2	35	77	160	68	61	74	131	607	4.0
IM3	40	25	138	24	38	36	178	478	3.1
Ave	43.8	51.2	185.7	40.7	37.7	53.8	121.6	534.5	3.52
XP1	26	0	256	83	42	92	238	736	4.8
XP2	12	66	246	124	76	106	39	669	4.4
XP3	11	13	115	42	23	80	90	373	2.5
Ave	***16.2***	26.2	205.6	82.7	46.9	92.7	122.4	592.5	3.90
OAve	53.3	52.8	191.1	53.1	41.0	62.4	158.6	612.1	4.03

Table 2A shows the individual group effort by activity as a percentage of the overall development effort.

Table 2A Individual group effort by activity as percentages

Group	*ReqsSpec %*	*Design %*	*Code %*	*Integr.& Test %*	*Review %*	*Repair %*	*Other %*	*Total %*
VM1	13	15	22	2	4	7	38	100
VM2	7	6	32	1	8	12	33	100
VM3	11	9	25	3	11	7	34	100
VM4	10	10	29	14	5	5	27	100
Ave	**10**	10	27	5	7	8	33	
EM1	11	13	34	9	1	10	22	100
EM2	8	16	25	8	3	14	26	100
EM3	14	8	39	8	6	6	19	100
EM4	17	5	19	20	9	5	26	100
Ave	**12**	10	29	11	5	9	23	
IM1	11	10	50	6	3	10	11	100
IM2	**6**	13	26	11	10	12	22	100
IM3	8	5	29	5	8	7	37	100
Ave	*8*	*9*	*35*	*7*	*7*	*10*	*23*	
XP1	3	0	35	11	6	12	32	100
XP2	2	10	37	18	11	16	6	100
XP3	3	3	31	11	6	21	24	100
Ave	***3***	4	34	14	8	17	21	
Min	2	0	19	1	1	5	6	
Max	17	16	50	20	11	21	38	
OAve	8.9	8.9	30.8	8.9	6.5	10.0	25.7	100
SD	4.3	4.1	7.3	5.2	2.9	4.3	8.7	

Table 3A provides the team-by-team results pertaining to the outputs of the early definition activities (requirements and top-level design).

Table 3A Metrics for requirements and high level design

Group	*Pages*	*Use cases*	*Screens*	*DB diagrams*	*DB Tables*
VM1	26	17	12	1	4
VM2	25	5	4	1	2
VM3	22	2	2	1	3
VM4	15	6	9	1	4
Ave	**22.0**	7.5	**6.8**	1.0	3.3
EM1	11	14	7	1	8
EM2	15	14	14	1	9
EM3	29	14	13	2	4
EM4	21	14	10	1	6
Ave	**19.0**	14.0	11.0	1.3	6.8
IM1	20	16	2	1	6
IM2	27	13	11	2	2
IM3	21	15	2	1	5
Ave	**22.7**	14.7	**5.0**	1.3	4.3
XP1	4	9	14	1	3
XP2	1	10	17	0	8
XP3	12	6	22	1	4
Ave	***5.7***	8.3	***17.7***	0.7	5.0
OAve	17.8	11.1	9.9	1.1	4.9

Table 4A, shows the design output per team in terms of the numbers of different diagrams.

Table 4A Design metrics

Group	*Overview diagrams*	*Class diagrams*	*Sequence diagrams*	*State diagrams*	*Other diagrams*	*Total*
VM1	1	4	2	0	3	10
VM2	1	4	2	0	0	7
VM3	1	1	3	14	3	22
VM4	1	1	2	7	1	12
Ave	1.0	2.5	2.3	5.3	1.8	**12.8**
EM1	1	1	3	2	1	8
EM2	2	1	4	0	2	9
EM3	1	2	1	0	2	6
EM4	1	2	7	0	1	11
Ave	1.3	1.5	3.8	0.5	1.5	8.5
IM1	1	7	5	0	2	15
IM2	4	1	14	0	1	20
IM3	2	3	4	0	3	12
Ave	2.3	3.7	**7.7**	0.0	2.0	**15.7**
XP1	2	0	0	0	0	2
XP2	0	0	0	0	1	1
XP3	1	0	0	0	1	2
Ave	1.0	0.0	***0.0***	0.0	0.7	***1.7***
OAve	1.4	1.9	3.4	1.6	1.5	9.8

Table 5A shows the product size delivered by each team in terms of the number of Java classes and the number of lines of code in a variety of programming languages.

Table 5A Performance metrics: Classes and lines of code

Group	*Java classes*	*Java*	*jsp*	*XML*	*Other*	*Total LOC*
VM1	10	905	1105	0	46	2056
VM2	7	648	427	12	45	1132
VM3	8	1300	2600	40	16	3956
VM4	9	1276	511	0	0	1787
Ave	8.5	**1032**	1161	13	27	**2233**
EM1	50	1665	1309	0	0	2974
EM2	9	1710	2133	0	130	3973
EM3	31	1254	2741	0	0	3995
EM4	17	1278	1429	0	17	2724
Ave	26.8	**1477**	1903	0	37	**3417**
IM1	42	3620	0	73	0	3693
IM2	10	289	1100	36	0	1425
IM3	10	1501	1666	0	42	3209
Ave	20.7	**1803**	922	36	14	**2776**
XP1	16	1849	2105	0	638	4592
XP2	38	6028	1229	2379	124	9760
XP3	29	6632	1652	583	0	8867
Ave	27.7	***4836***	1662	987	254	***7740***
OAve	20.4	2140	1429	223	76	3867

Table 6A shows the computed metrics derived from combining different sets of data.

Table 6A Derived metrics

Group	*Java LOC / class*	*Classes/ PM*	*LOC / PM*	*RS&HD pages / PM*	*LOC pages / PM*	*Total pages / PM*
VM1	91	3.0	607	22.1	12.1	34.3
VM2	93	1.2	190	12.5	3.8	16.3
VM3	163	1.3	654	19.6	13.1	32.7
VM4	142	2.1	418	18.2	8.4	26.6
Ave	122	1.7	**467**	**18.1**	**9.3**	27.5
EM1	33	10.3	610	18.9	12.2	31.1
EM2	190	3.0	1306	15.0	26.1	41.1
EM3	40	8.0	1033	23.0	20.7	43.7
EM4	75	6.4	1021	29.4	20.4	49.8
Ave	85	7.4	992	**21.6**	19.9	41.4
IM1	86	12.3	1081	21.1	21.6	42.7
IM2	29	2.5	357	13.8	7.1	20.9
IM3	150	3.2	1022	16.4	20.4	36.8
Ave	88	5.9	820	**17.1**	16.4	33.5
XP1	116	3.3	948	5.7	19.0	24.6
XP2	159	8.6	2219	3.0	44.4	47.3
XP3	229	11.8	3618	5.3	72.4	77.7
Ave	168	7.1	***2262***	***4.6***	***45.2***	49.9
OAve	114	5.5	1077	16.0	21.5	37.5

Table 7A shows the quality ratings achieved by each team, using three of the ISO9126 criteria.

Table 7A Product quality assessment on scale 0-10

Group	*Functionality*	*Usability*	*Maintainability*	*Ave*
VM1	9	8	8	8.5
VM2	6	7	9	7.2
VM3	n/a	n/a	n/a	n/a
VM4	9	7	10	8.7
Ave	7.9	7.4	8.9	8.1
EM1	6	7	7	6.7
EM2	9	8	8	8.2
EM3	9	7	9	8.3
EM4	8	7	8	7.6
Ave	7.9	7.3	7.9	7.7
IM1	6	8	7	7.0
IM2	10	10	10	10.0
IM3	9	7	5	7.0
Ave	8.3	8.3	7.3	8.0
XP1	8	7	7	7.3
XP2	9	8	9	8.7
XP3	7	8	7	7.3
Ave	8.0	7.7	7.7	7.8
OAve	8.0	7.7	8.0	7.9