

COMPUTER SYSTEM DEVELOPMENT: PROBLEMS EXPERIENCED IN THE USE OF INCREMENTAL DELIVERY

F. J. Redmill

British Telecom International, UK

ABSTRACT

Incremental delivery offers advantages over the waterfall development model. This paper, based on experience, confirms some advantages and describes a number of problems which developers and project managers must deal with.

Keywords. Project management; software engineering; incremental delivery.

INTRODUCTION

The "big bang" or monolithic approach to software development has not led to complete customer satisfaction. Too often, systems do not meet customers' requirements and are either abandoned or subjected to considerable change. In one extreme example, [US GOV '79], of 9 federal software projects in the USA, costing \$6.8 million, the following results were achieved: 47% of the software (\$3.2 million) was paid for but never delivered, 29% (\$2 million) was delivered but never used, 19% (\$1.3 million) was abandoned or reworked after going into service, 3% (\$0.2 million) was used after change, and only 2% (\$0.1 million) was used as delivered.

The heart of the problem is the need for change. Yet, the great strength of software is its flexibility, its ease of change, and that is why we use it. So why does the problem arise? Too much change is unexpected, inconvenient, uneconomic, difficult to achieve, and resource-consuming to carry out. There is a need:

- (i) To control change;
- (ii) To minimise change due to earlier failure;
- (iii) To recognise the need for change early when it is least expensive.

Change is best controlled if the project is clearly defined and fits into a consistent and well-coordinated strategy. Then, new users are less likely to appear late in the project or when the system is already in service. Change due to failure is currently minimised by project management, software engineering, and quality assurance. Recognising early the need for change may be achieved by continuous user involvement throughout the project, by the use of non-monolithic development models, and, in particular, by incremental delivery which offers users an early opportunity to use the system and to review their needs in the light of their experience of it.

The fault is not always with the big bang approach: frequently this takes the blame for poor systems analysis, lack of user involvement, or inferior software engineering in the development process. Nevertheless, it is now generally accepted that there are advantages to be gained from the use of non-monolithic development models. But, along with advantages come difficulties. Based on the use of incremental delivery over a number of years and in a number of projects, this paper describes the difficulties which may accompany its application. In many cases, solutions are either discussed or are implicit in the descriptions of the problems.

DEFINITIONS

In the traditional monolithic, or big bang, approach to development, all the detail of each stage is (ideally) completed before the next stage is begun. This involves strict adherence to the waterfall development model, and no part of the system is operational until the end of the project, when the whole system is completed, tested against the original specification, and brought into service.

In the incremental approach, functional modules, or increments of the system are identified and developed while completion of some or all stages is deferred. Thus, increments become operational, and may be delivered and brought into service, before the whole system is complete. Indeed, the system is constructed by the successive addition of new increments (and the changes to existing increments). There are a number of non-monolithic development models, and these have been described [GRAHAM '89]. Incremental development may be employed without delivery, but a great advantage, that of feedback, is only gained when the increments are delivered, brought into service, and used.

PREREQUISITES TO EFFECTIVE DEVELOPMENT

No development model offers a guarantee of success. The advantages and disadvantages of incremental delivery (or any other method) can only be discussed meaningfully if it is assumed that they are not distorted by other, avoidable forces. A system of high quality can be achieved only if there is the convergence of a number of influences. The important influences are briefly commented on below in four categories, each being extremely broad and containing a large number of topics which, in a different context, would be considered individually.

Context and Bounds of the System

It is astonishing how many systems are developed without any participant in the project, either user or developer, knowing the business strategy into which the system must fit, the purpose of the system within that strategy, or the relationship of the system to other parts of the business. If these matters are not defined, there can be no certain boundary to the system, and, thus, no clear statement of who the system's users will be; by implication, no one is excluded from using the system. Consequently, throughout the project, and even after installation, new users are likely to demand access to the system as well as new and

changed facilities. With a strategy and business objectives for the system, the initial specification and the specifications for changes can be made to remain within the defined boundary, and change can be controlled.

Specification

Only a good specification provides designers with the information to size the system, choose the most appropriate hardware and system software, estimate costs and timescales for the project, and arrive at the optimum modular design for the applications software. It may be tempting to believe that incremental development and delivery obviate the need for a good initial specification, but few businesses are prepared to authorise a project if costs and timescales have not been defined with some degree of confidence. And few systems will be of optimum design if the initial designers had no idea of what was to come later. Indeed, it is well accepted that changes to design increase complexity and decrease efficiency; and incremental delivery does not avoid this.

Software Engineering

No development model obviates the need for defined standards, comprehensive guidelines, well-understood procedures, engineering principles, proven tools and techniques, independent verification and validation, disciplined configuration management, strict quality assurance, and sound management. "Software engineering" is a discipline which combines all these, and a lack of any of them jeopardises the quality of the system.

Project Management

No matter how keen and competent the individuals of a development team may be, their cohesion, direction, and achievement depends on good project management. This ensures that all work is planned and that reviews of completed work provide feedback which influences the plans for future work. It includes getting the organisation right, recruiting the right staff, attention to staff development and training, active participation and interest, creating a reporting structure and communication opportunities, leadership, planning, monitoring progress, use of feedback, ensuring that software engineering practice is adhered to, and project control.

ADVANTAGES OF INCREMENTAL DELIVERY

It is not intended to discuss here the theoretical advantages of incremental delivery, for the proponents of non-monolithic development models (for example, GILB '85, KRZANIK '86) have already set them out. However, it is worth listing those which experience has confirmed.

- (i) It provides the customer with a working system (with a limited number of functions) much earlier than otherwise;
- (ii) Early delivery gives the customer confidence in the development team, particularly if the functions perform correctly first time;
- (iii) An early in-service system boosts the morale of the development team;
- (iv) Feedback from users on delivered functions, and consequent revisions of future work help to get the final product right.

Delivering a system incrementally, however, incurs problems, and these are the subject of this paper.

DIFFICULTIES

Incremental delivery gives rise to a range of difficulties which affect developers, users, and project management. The following sub-sections describe them, though, with so many issues, only brief discussions are possible.

Bottom-up Management

As stated earlier, top-down control of development is best achieved by having a strategy into which the target system fits and which places well-defined bounds on the target system.

However, incremental delivery can provide a framework for bottom-up management unless it is carefully controlled. The requests for change are generated by system users, and, as already shown, the resulting work can cause the project to go out of control. In an attempt to accelerate the flow of work, the procedures for specifying changes may be left to junior staff. This has the disadvantage that senior management may be short-circuited, and the result that the changes may help individual users but not the business as a whole, i.e., that efficiency is achieved at the expense of effectiveness. To avoid this, there not only needs to be a strategy and bounds on the

project, but also a formal method for ensuring that all changes are within them.

Specification

Users, as well as developers, are normally busy. Unfortunately, preparing a specification is often not their only, or even their primary function. Knowing that the system will be developed and delivered in increments, they may decide to save time during the specification stage of the project by specifying the system incrementally. While incremental delivery implies a regular review of the users' requirements, incremental specification has been found not to be satisfactory. It is a distinct disadvantage not to have a comprehensive specification prior to design. However, it is not usually a problem if the original specification provides information on basic facilities and their boundaries, with further detail to follow. Producing a good specification prior to design is not easy (REDMILL '87), but customers should be encouraged to provide one, and assisted if necessary. The penalties for not having one are significant in estimating and design.

Estimating

Without an understanding of the full range of the users' requirements, it is not possible to estimate the project's resources, cost, and time. It is true that development can proceed without estimates, but no well-run business will commit itself to a project of unknown cost and completion date. Nor would a self-respecting project manager permit aimless development. However, efficient estimating depends not only on basic information, but also on constantly comparing the expenditure of resources against estimates. This provides evidence on which to base future estimates, and depends on the accurate recording of the use of time and on good project management.

If the developers are forced to produce estimates in the absence of a full specification, they find themselves trapped. Low, optimistic estimates usually lead to premature exhaustion of funds, the exposure of the poor estimation, and the need for reauthorisation of the project. Senior managers may comprehend only the request for an increase in budget, and not the difficulties which caused it. On the other hand, estimates which are seen to make allowance for unspecified requirements, and to include experience-based tolerances, are often seen by a business as unjustifiable. Deficiencies in the specification are overlooked, and it is assumed that developers are allowing for their own

inadequacies. Estimation is difficult at the best of times. Doing it without a specification requires prayer.

Design

Even with a comprehensive specification, it is often impossible to achieve the most efficient design, the best that can be done being to introduce as much flexibility as possible to allow for the expansion and change which are to come. With only a partial specification, there may be difficulty in choosing the optimal hardware and system software. Moreover, the functionality, reliability, maintainability, and, ultimately, the expandability of the final system are likely to suffer, for later design modifications lead to increased complexity, decreased efficiency, and increased difficulty, effort, and cost in achieving desired results. For example, timing within the system cannot be guaranteed, and it may be difficult to meet the required times of response to users in transaction-processing systems, or of events in real-time systems. In critical systems, this can be a major problem.

Development System

Whereas the big bang approach allows the development system ultimately to go into service as the target system, incremental delivery does not. Having a separate development system is a maintenance asset for the life of the target system, but its cost must be justified when the case for the project is prepared.

Configuration Management

For incremental delivery, configuration management is a great deal more complex than for the monolithic approach, and it requires more effort and care. In monolithic development, the system is configured at the time of installation and, thereafter, changes are made as the system is modified. In incremental delivery, there are a number of versions of the system at any given time. Maintaining a schedule of regular deliveries and meeting users' requests for high-priority changes do not allow the development of one increment to be delayed until the previous one is in service. Thus, if Version 1 is the system in service, Version 2 is the next delivery, Version 3 the delivery after that, etc, then all must be based on Version 1. When Version 2 goes into service, all later deliveries must be based on this, so the configurations of all other versions must be changed accordingly. All new and

changed software must be incorporated into later versions and their documentation brought up-to-date. It must also be established how the work already done within these versions may be affected by the new software. Achieving these aims requires not only highly-disciplined (and, preferably, automated) configuration management, but also careful planning to ensure that successive deliveries do not contain changes to the same module.

Revalidation

When a working system has been changed, it is proper practice to revalidate it under working conditions, before returning it to service. Revalidation can be a lengthy business if carried out adequately, and this is difficult for users to accept. Once they have become accustomed to using the system, they do not want to be without it. Indeed, they should not be without it if it is to replace their earlier method of working. A number of points are worth commenting on.

- (i) **Frequency of Deliveries.** A second delivery cannot be made within the time required for revalidation. Indeed, considering the effort and time required for revalidation, deliveries need to be carefully planned and well spaced in order to optimise the balance between the provision of new functions and facilities and the time for their development. The idea of a delivery per week has not been found to be practical.
- (ii) **Time.** Because of (i) above, a greater proportion of time is spent in revalidation when incremental delivery is employed. If full revalidation of the system precedes each delivery (and this is essential for critical systems), the time and effort required increases as development proceeds. If deliveries are frequent, this places a great load on the developers' resources, and there is a danger that there will be attempts to make savings by only revalidating selected parts of the system on each occasion.
- (iii) **Compromise.** In order not to deprive the users of the system for longer than it takes to install a new software version, revalidation may have to be carried out on the development system. This may not test the system under operational conditions and

should not be permitted in the case of a safety-related (or otherwise critical) system.

Requests for Change

Experience of the system leads to requests for change to the existing functions and facilities and for previously unspecified ones. If each request is not specified completely, correctly, and unambiguously, there exists the danger of abortive work, later disagreement, and quality-related costs. Yet, the customer's organisation may not be equipped for continuous specification production. Having prepared the original specification (or not, as the case may be), the staff involved may have returned to other duties. Development on the basis of verbal specification is dangerous. The customer has a duty to provide precise specifications, and the developers have a duty not to schedule, estimate, or carry out the work in their absence.

Nature of Deliveries

Deliveries should be planned. Users should be involved in the planning so that they receive their highest-priority requirements first, understand the reasons why they can't get everything they want at the same time, and know what to expect and when to expect it. This implies that the content of each delivery must be precisely specified, the effort necessary to achieve it estimated, and the development and testing controlled so that targets are met. An advantage of incremental delivery is that the developers gain prestige by regularly supplying new facilities to the users. However, prestige can be lost if promises are not kept.

The first delivery is necessarily based on work defined in the original specification. However, each delivery gives rise to demands from the users for change or new features. And, often, these are accorded a very high priority. Typically, therefore, a delivery consists of both new features and changes to the existing system. Again it is emphasised that careful planning, based on full specifications for all work, is essential.

Prioritising and Scheduling Work

As deliveries are made, users are often more anxious to improve the existing functions and facilities than they are to receive new ones. Thus, as new requirements arise, the project

manager needs to ensure a regular reprioritisation of features to be provided. This should involve both users and developers, and achieves a number of objectives.

- (i) It allows immediate use of feedback for the improvement of the system;
- (ii) It ensures that each delivery is optimal, from the users' point of view;
- (iii) It enables requirements which have been superseded by requests for change to be acknowledged as being obsolete, thus helping the developers to avoid abortive work;
- (iv) It brings key members of the project (including users and developers) together so that they can understand each other's needs and problems, reach compromises when necessary, and agree on common goals.

It is assumed here that the project is managed effectively and that the level and quality of user representation is such that there is an adequate balance between satisfying the genuine needs of end users and meeting the objectives of the business.

From the developers' point of view, new requirements and reprioritisation results in rescheduling the contents of future deliveries. And this needs to be done without undue disappointment to end users, who are usually not the final arbiters of priority. Tact is needed by the developers, who should also build a certain flexibility into the plans for future deliveries.

Controlling Change

A major advantage of incremental delivery is that it allows early feedback from users of the system. This leads to change, which is intended to make the system more suitable for the users' needs. However, as the previous paragraphs have shown, change needs to be controlled. If it is not, large numbers of requests for change (often trivial, sometimes duplicated, and sometimes one superseding an earlier one), may cause project timescales and costs to spiral. Control needs to be applied by having formal methods for the analysis and adjudication of requests for change, objectives against which to judge the value of the changes, the linking of all changes to the original specification, prioritisation of work,

and careful structuring of the development team.

Development Team Structure

In spite of reprioritising work so as to accommodate the users' requests for change, the management of a business may judge the success of the development team by their progress against the original specification and the original plans. If the estimates and plans for the project are based entirely on the original specification, the project will be seen to slip as soon as requests for change begin to arrive. Yet, a major advantage of incremental delivery is that it attracts early requests for change. So, plans need to allow for the future diversion of effort to meeting requests for change. One way of doing this is to plan normally up to the first delivery, and then to plan for only a part of the team to continue development according to the original specification. The optimum balance is difficult to achieve until the extent of the requests for change is discovered. However, unless this separation is made, either the original specification is adhered to and the benefit of incremental delivery is lost, or requests for change dominate the work and control of the project is lost (at least, as seen by senior management).

Having a fixed number of staff on requests for change (by agreement with the project manager and users) facilitates accuracy of planning the delivery of new requirements and encourages the users to prioritise their requirements. Yet, in some cases, this is not the whole answer. If new work has a higher priority than the requirements in the original specification, the new work should take precedence. What suffers then is the image of the development team, unless the senior management in the business fully understand incremental delivery and its implications.

Acceptance Testing

Acceptance testing requires the customer's effort in testing the system in various ways, including under operational conditions. As in the case of system revalidation, the users do not want to lose the use of the system, except for the short periods necessary for delivering a new version of the software. Thus, in some cases, there is pressure from the users to carry out acceptance testing on the development system.

There is a tendency, however, for users to perceive acceptance testing as being completed

when the first delivery is made. After that, deliveries are seen as entirely the responsibility of the developers. Preparing an acceptance test specification for every request for change is not trivial. A customer may not have adequate staff for this, or may want to avoid the task.

In all cases, the developers must ensure that there exists a precise specification for the functions or changes being delivered, and that this contains clear acceptance criteria (as should the original specification): if the users will not draft an acceptance-test specification, the developers must be confident that system revalidation provides an adequate level of confidence, not only in the new software, but also in its integration with the existing operational system.

Maintenance

Traditionally, all changes made to a system after bringing it into service have been designated as maintenance. When there has been a need or desire to distinguish between fault correction and externally-imposed change, the categories "corrective maintenance", "perfective maintenance", and "adaptive maintenance" have been used [SWANSON '76]. However, separate costs have not often been derived for these categories, and, consequently, a great deal of development has been carried out under the guise of maintenance.

Swanson's categories of maintenance are suitable for a system whose development is considered to have been completed, which is wholly in service, and which is supported by a dedicated "maintenance" team. However, while incremental development is in progress, maintenance applies only to the correction of faults found in increments already delivered. All other work is development. Accurate project accounting depends on making this distinction and on being disciplined in the appropriate recording of time spent and costs incurred.

Development Team Image

A business can easily get a very distorted impression of the progress of a project, against both time and cost, unless the developers take care to record and publicise true statistics. When it comes down to it, a business is interested in a system becoming functional rather than in the details of its development. Senior management see the system as being developed within a project, and they therefore assume that all the project's

forecasts (budget, resources, time scale) apply to the development of the complete system. However, in incremental delivery, there are 3 categories of work.

- (i) Development according to the original specification, on which resource, cost, and time estimates were based.
- (ii) Development according to requests for change (RFC) made after the specification had been agreed and the project budget authorised. Typically, RFCs are for new or changed facilities not previously specified, or for adaptive maintenance.
- (iii) Corrective maintenance.

The first and third of these come under the project budget. Yet, the second may give rise to a significant proportion (sometimes, the majority) of the development team's effort, once the early deliveries have been made and are being used. Unless the development team keeps accurate records, to show that much of the work being done is outside of the project budget, it may be seen as not meeting its targets.

Project Management

With proper use of the waterfall model, only one stage of a project can be operative at a time. The project team thus works to one project plan and one stage plan, the flow of events is sequential, and the complication of parallelism does not occur except in scheduling tasks within a stage.

With incremental delivery, this is not the case. When the first increment has been developed and tested, it enters first the Installation Stage and then the Operation (and Maintenance) Stage. Whereas subsequent deliveries of software may not formally pass through the Installation Stage, thereafter there will always be 2 concurrent operative stages of the project, the Operation Stage of the existing system and the Development Stage of the future deliveries. As seen above, this complicates planning and configuration management, as well as project reporting and control. In addition, there is the added problem of co-ordinating the feedback from users, arranging for the regular prioritisation of requirements, handling the statistics concerning changes, and recording the effort and cost involved in meeting new requirements. The latter is crucial, for project funding depends on it.

Being in two project stages concurrently is unavoidable, and the result is increased complexity in project management.

Budget

Difficulties with the project budget have been mentioned in a number of the above paragraphs. The fact is that when incremental delivery is used, this needs to be taken into account when the project budget is authorised. Just as the original specification is not a reliable guide to the eventual set of requirements, so a budget based on that specification is unlikely to be adequate for the project. Both customers and developers must endeavour to ensure that all effort is effectively used, for the freedom inherent in incremental delivery should not be abused, but the principle of a changing budget needs to be accepted as fundamental. However, this does not mean carte blanche. A company still has to ensure that a system's cost does not exceed its value to the business. Thus, initial estimating must show with a high degree of confidence that the total required budget will not exceed a given maximum. A good initial specification, a good understanding of the system's functions and objectives, and a consideration of the effort likely to be needed for changes are necessary for this.

DISCUSSION AND CONCLUSIONS

This paper has reported on experience in the use of the incremental delivery of computer systems. First, it pointed out that for any development model to be successful, a number of conditions need to be fulfilled. These include good strategic planning, specification, software engineering, and project management; and each of these contains a wide range of topics. Next, the paper briefly confirmed some of the advantages claimed for incremental delivery.

However, the principal aim of the paper was to describe a number of problems, or potential problems, which accompany incremental delivery and which must be overcome by developers and project managers. These were explained under 17 headings. Many of the problems are matters of increased complexity rather than technical obstacles, and require disciplined management rather than new tools or techniques. They are issues which must be prepared for and dealt with rather than avoided, and this paper is intended to alert developers and project managers to them. In many cases, solutions were touched on. In others,

awareness and understanding of the problem should be sufficient to stimulate the necessary management reactions or precautions.

In spite of these issues, incremental delivery is recommended, particularly for large projects. The most important point to note is that control is difficult but essential. However, incremental delivery does not altogether replace the waterfall model. The latter provides a sound, straightforward set of principles which should be adhered to in the development of each increment.

Finally, it should be said that the experience reported in this paper is derived from carrying out in-house development projects. If incremental delivery is to be used in contracted-out projects, even greater care would need to be taken in controlling requests for change to the system. Each request (except corrections) would be a variation to the specification and would need to be costed and contracted for. The project could become a "time and materials" contract with an uncontrolled specification. However, it is felt that most of the points made in this paper would still be relevant.

REFERENCES

- [GILB '85] Gilb T: Evolutionary Delivery versus the "Waterfall Model". ACM Sigsoft Software Engineering Notes, Vol. 10, No. 3, July 1985.
- [GRAHAM '89] Graham D R: Incremental Development: Review of Non-monolithic Life-cycle Development Models. Information and Software Technology, Vol. 31, No. 1, Jan/Feb 1989.
- [KRZANIK '86] Krzanik L: Software Development Standards for Non-monolithic Project Management Strategies: Experience, New Suggestions and Conclusions. 1st IFAC Workshop on Experience with the Management of Software Projects. Heidelberg, FRG, May 1986.
- [REDMILL '87] Difficulties of Specifying Users' Requirements for Computer Systems and Methods of Mitigating them. British Telecommunications Engineering, Vol.6, Part 1, April 1987.
- [SWANSON '76] Swanson E B: The Dimensions of Software Maintenance. Proceedings of the 2nd International Conference on Software Engineering. IEEE Computer Society, October 1976.
- [US GOV '79] US Government Accounting Office Report, 1979. As reported in Pressman R S: Software Engineering. McGraw-Hill International, 1987.