C. Wohlin, "Managing Software Quality through Incremental Development and Certification", In Building Quality into Software, pp. 187-202, edited by M. Ross, C. A. Brebbia, G. Staples and J. Stapleton, Computational Mechanics Publications, Southampton, United Kingdom, 1994.

Managing Software Quality through Incremental Development and Certification[1]

C. Wohlin
Department of Communication Systems, Lund Institute of Technology, Lund University, Box 118, S-221 00 Lund, Sweden

ABSTRACT

A major problem in software development is the contradiction between on-time delivery and reliability of the software. Improvements are needed to support on-time delivery without sacrificing the reliability requirement.

Incremental development and certification will allow for a better combination of on-time delivery and reliability fulfilment. A method proposing an optimum order of development and certification of increments is presented. The method is based on mapping the requirements into increments, where the mapping must be made so that an optimum order can be determined. The order of increments will be based on risk and usage of the increments. It is shown that the proposed method is superior to the waterfall approach and to a random development and certification order of the increments. It is concluded that incremental development and certification does help the software manager to fulfil the on-time and reliability requirements at the same time.

INTRODUCTION

The basis for managing software quality is control throughout the development. This can be achieved by incremental development, since the basic idea with incremental development is to implement an executable part of the system. The incremental approach allows for reliability certification of each increment and in particular the software system may grow and thus it is possible to certify the cumulative growing system. This approach is advocated in Cleanroom Software Engineering (eg. Mills [1]).

1. This work is supported by National Board for Industrial and Technical Development, Sweden, reference Dnr: 93-2850.

The real benefit from the incremental approach can only be enjoyed if the increments are developed in an appropriate order. The order must support managers so that the software system meets the reliability requirements in as short time as possible. It is possible to find such an order and the objective of the paper is to present a method, which give the best order to fulfil both reliability and time constraints. The method supports software managers in their task to meet the quality objectives.

The key issues in the method are to identify suitable increments and to determine a suitable order of the development of increments. The latter is based on requirements regarding critical parts in the systems, overall software reliability requirements and an objective to minimize the time for development of the software product.

The paper contains some introductory sections, which introduces some different areas independently. First some principles in Cleanroom Software Engineering are presented to introduce the context in which incremental development is suggested. Cleanroom is, however, not a prerequisite to use incremental development. The relationship between time constraints and reliability of the software product is then discussed, after which some different development approaches are discussed briefly. This discussion is followed by an introduction to reliability certification.

The introductory sections are the basis for the sections in which the areas are combined. The development approaches are elaborated one by one in the light of reliability certification. The elaboration indicates the benefits with increments in the development, hence a method for determining a suitable order to develop the increments is proposed. The proposal is then compared in terms of time to delivery, both with a traditional waterfall model as well as with an incremental approach without the proposed scheme for determining the best order to develop increments. Finally, some conclusions are presented.

PRINCIPLES IN CLEANROOM SOFTWARE ENGINEERING

Introduction

Cleanroom Software Engineering, (eg. Mills [1]), has shown that it is possible to improve software quality and at the same time improve the productivity.

The Cleanroom methodology is based on the philosophy that it is possible to develop zero defect software, though it may be hard to prove. The overall principle in developing software systems using Cleanroom is to remove defects in the same development phase as they are introduced, instead of waiting for an executable code representation of the system to perform tests and defect removal on. Cleanroom may be seen as consisting of a number of principles which forms a basis for the overall philosophy.

The following principles are emphasized in Cleanroom:

1. Intellectual control of the software development.
2. Team responsibility of the work.
3. Process driven development.
4. Incremental development.
5. High level specification and design.
6. Stepwise refinement and rigorous verification.
7. Certification of software reliability.

These items are a collection of aspects stressed in the Cleanroom literature, but they still allow for application of different techniques. If these views are considered as Cleanroom, then several different techniques may be applied still fulfilling the objectives of Cleanroom, but some more specific techniques can also be found in the literature.

Box Structures (eg. Mills [2]) is proposed for specification and design of the software, Stepwise Refinement and Functional Verification (eg. Linger [3]) are methods for implementing code in small steps and verifying them mathematically, Statistical Usage Testing (eg. Cobb [4]) describes how the certification is to be done in Cleanroom. It is proposed that the usage shall be modelled with a plain Markov chain, (eg. Whittaker [5]), hence allowing for generation of test cases according to the anticipated usage. The reliability model proposed is presented in (eg. Currit [6]).

In particular, no specific method to support the principle of incremental development is explicitly presented within Cleanroom. The objective of this paper is to propose such a method to better enjoy the benefits of incremental development, see item 4, and certification, see item 7.

TIME TO DELIVERY VERSUS RELIABILITY

The discussion is mainly concerned with two quality attributes: time to delivery and reliability. These two often seem to contradict each other. The procurer of a software product may get the product on time, but the software has not been thoroughly verified before the release due to delays. The delays often means taking short-cuts in the verification and validation of the system before releasing the software. In other words, poor software in terms of reliability is delivered due to time constraints. The testing is not carried out as it ought to; it may even happen that untested software hits the field.

The requirement on a specific release date often overrules other quality objectives as for example reliability. This is of course unacceptable, but all too

common. Two major improvements are needed to come to term with the problem:

1. Management maturity
   This is a non-technical issue, which refers to that management must be able and prepared to give a realistic estimate of the time to delivery. This includes taking other quality requirements into account, hence obtaining a realistic combination of time to delivery and reliability. This improvement is not discussed here, even though it is very important.

2. Process improvement
   This item is used as a collective term for the technical improvements that can be made, but in particular it refers to the improvements that can be made in the actual development process. This process must of course be supported by methods and tools. These issues are, however, not discussed in this paper. The objective is, as stated earlier, to suggest a method to improve the actual development process to enhance the reliability of the software without prolonging the development phase. The key issue is to use the available development time in an optimum way.

DEVELOPMENT APPROACHES

Waterfall model

The waterfall model for software development is well-known and does not need an explanation, but to emphasis the important properties of the model which influence the conclusions of the paper a brief presentation is included. The waterfall model is described in more detail elsewhere (eg. Boehm [7]).

The model is based on a flow from idea (or requirement specification) to maintenance of the software, through several development steps, for example design, coding and testing. The actual steps in the waterfall model may be different depending on development environment and development process formulation. The important issue in our context is that the model assumes that the requirements are written for the whole system to be developed, then the system is taken through the model step by step. The system is designed, coded and tested in a series of activities. The system may be broken down into subsystems and modules to enhance the development, but the system goes through the model more or less synchronized.

The system is hence not tested as parts, but reaches the system test all at once. This is at least the basis of the model, in practice there may be some deviations, due to problems with some parts of the system or other organizational dependent reasons.

The waterfall model does assume that the complete system is developed after which it is tested. In particular, this implies that development and testing are not parallel activities.

Spiral model

The spiral model is an improvement and refinement of the waterfall model (eg. Boehm [7]). The model is based on a risk driven approach and supports prototyping. The background of the model is the insight that it is not feasible to develop software systems as proposed in the waterfall model. The development must be an iterative procedure.

The model can be viewed as consisting of several waterfall models performed after each other, where the output from one waterfall model is a prototype of the system (or part of the system) which can be shown to the procurer of the system. The objective with this approach is to allow for a better communication between the developer and the procurer. The software is shown to the procurer at different points during the development hence allowing for better visibility of the product being developed.

The spiral model is a risk driven approach. It emphasizes to start with concentrating on high risk elements, for example the user interface may be judged as constituting a large risk. If this is the case the development starts with constructing a prototype of the user interface, which then is shown to the procurer. The evaluation of the prototype results in improved requirements, which work as input to the next development cycle.

Incremental model

The objective with this model is to identify parts which can be developed from specification to executable code. The development of an increment may follow either a waterfall model or a spiral approach. Incremental development means dividing the requirements into suitable parts during the specification allowing for independent development of the different increments.

The design and coding of one increment are followed by testing of that increment, which makes it possible for the developers to start implementing the next increment while the testers validate, verify or certify the first developed increment. Here, it is assumed that development and testing are performed by different teams, which also is one of the principles in Cleanroom.

The incremental approach hence allows for a good deal of parallelism between development and testing. The benefit from this parallelism is not only the possibility to work in parallel, but also that the testers really start testing the software to be delivered at an early stage. This property will be essential in the discussion below, since it supports early reliability control through certification of increments.

Two major questions arise when discussing incremental development:

- How are increments identified from a requirement specification?
- In which order ought the increments to be developed?

The first question is very hard to answer from a general point of view. It is important that the persons specifying the system have a good knowledge and experience of the application domain. The formulation of suitable increments is still a creative process. Some guidelines can be found when answering the second question, which is further discussed below.

Summary development approaches

The spiral model may seem similar to incremental development, but there are fundamental differences. Incremental development emphasizes identification of system parts which may be implemented from specification to testing and which are expected to be parts in the system to be delivered. In the spiral model the objective is to develop prototypes which can be shown to the procurer to form a basis for requirement refinement and as a means for better understanding between developer and procurer.

Both these approaches have their benefits, but only the incremental approach allows for development and certification in parallel of the system parts which actually shall be delivered. The prototypes may be important, but they do not support continuous statistical quality control of the software to be delivered, and hence it is difficult to be in control of the software reliability at the same time as fulfilling the requirements on delivery time.

Therefore, it is concluded that incremental development is superior to the spiral model in terms of quality control. The spiral model will not be used any further in the comparison since it requires too many assumptions concerning what the prototypes actually are. This difficulty makes it extremely hard to compare the spiral model with the incremental approach in a quantitative manner, which is the objective below.

The waterfall model will however be compared with the incremental approach concerning ability to deliver a software product with a specific reliability requirement.

CERTIFICATION

Certification is the control of the quality fulfilment, for example to certify that a specific reliability has been achieved. The basis for reliability certification is usage testing. Reliability certification consists of two main constituents:

- Usage description, which includes both a usage model and a usage profile. The model must describe the usage as it is expected in the operational phase. This means describing a user of the system in terms of states of the user and actions a user of the software can perform. The profile describes the relative frequencies to perform certain actions, i.e. the actions are assigned probabilities. The usage description is used to select test cases

which resemble the anticipated usage of the software when in operation. Some different techniques to model usage are discussed in Whittaker [5], Runeson [8] and Musa [9].

- Reliability estimation and prediction procedure, which includes different techniques to perform estimation and prediction. Several different methods and models are presented in the literature. The methods range from sampling techniques (eg. Parnas [10]) to software reliability growth models (eg. Currit [6]) via hypothesis testing methods (eg. Musa [11]). The choice of the best method and model may differ between different development environment.

The objective with this presentation is not to present reliability certification techniques in any depth, but to show that the techniques are available and to present the constituents briefly. Therefore it can be stated that it is possible to estimate and predict the reliability or the mean time between failures (MTBF) applying usage testing and reliability certification.

It will hereafter be assumed that it is possible to model the usage of a system or an increment, generate test cases according to the expected usage, collect failure data, estimate and predict the MTBF. This assumption is essential in the presentation below.

CERTIFICATION WITH DIFFERENT DEVELOPMENT APPROACHES

In the waterfall model, certification can only be applied during system test. This is the first time when it is possible to generate test cases and test the software according to the anticipated usage. It is, however, not possible to generate test cases according to the usage for units and separate functions.

The units are mostly not seen directly by the user hence it is difficult to certify them from a usage point of view. Certification of units or components are, however, an important research area if reuse is to be a major breakthrough for software systems (eg. Poore [12] and Wohlin [13]), but it will probably not be an applicable method during unit testing in a particular project. It will rather be a method to certify components which shall be stored in a repository for future reuse.

Reliability certification can not be applied during functional testing, since the objective of functional testing is mainly to assure that a particular function is correctly implemented. Therefore not allowing for certification from a statistical viewpoint.

The reasoning above for the waterfall model can be applied to the spiral model as well, since the steps in the spiral model only produces prototypes which shall not be delivered. They can hence not be tested according to a usage

profile, therefore the first time a reliability certification is feasible is during system test in the spiral approach as well.

In the incremental model, the reliability certification can be applied on each increment, since the definition of an increment is that it is an executable part of the system. The first certification is performed on the first increment, while the second certification is done on increment one and two together and so on. The system grows increment by increment until the complete system finally can be certified. A usage description must be formulated on an incremental level to allow for the certification. The ability to certify the system incrementally is one of the major advantages with incremental development.

AN OPTIMUM ORDER FOR INCREMENTS

The introduction of incremental development will probably cause some initial problems, since it requires a cultural change in an organization. This may mean that special considerations must be taken when choosing the order of increments in a first project. Some special considerations may be necessary in other projects as well, but the objective here is to present the best order without considering the special cases which may be due to, for example political reasons, staffing problems or available test equipment.

The method presented below gives an optimum order without taking any special conditions into consideration. This is the only way to determine a suitable order, it is then up to the user of the proposed method to adopt it and apply it with any special needs taken into account.

The method means dividing the increments into three classes:

1. High risk increments

   The increments which are considered as high risk increments must be developed first, since the first developed increments will obtain most testing in the incremental model. The high risk increments will also be the increments on which the reliability requirements are the highest. Therefore, it is necessary to start with the high risk increments.

2. High usage increments

   The next class is the high usage increments. These are important since the high usage means that the reliability of the increments must be high to fulfil the overall reliability requirement.

3. Low usage increments

   The low usage increments are not that important from a reliability point of view since they are seldom used and their contribution to the overall system reliability is quite small.

These classes also support the specifiers in their task. In their work they must try to collect as homogeneous increments as possible, hence distributing the requirements on suitable increments to obtain a structure which supports the classes defined above. The definitions of these classes must act as guidelines for the specifiers when identifying increments.

TIME TO DELIVERY WITH THE DIFFERENT APPROACHES

Introduction

The actual time to delivery is a complex function, which here is regarded as constituting of development time and certification time. The development time is assumed to take a certain amount of time (in terms of calendar time), while the certification time primarily is a function of the initial reliability of the software as it is delivered to the certification team, the reliability growth during certification and the reliability requirement.

Due to the number of different situations that can occur, for example varying the parameters described in the previous paragraph and the opportunity to develop increments in parallel or not, it is not easy to find a general solution to when the incremental model is superior to the waterfall model. Therefore a intuitive explanation is first given. An example is then introduced to highlight how the incremental model can be compared with the waterfall model and also to show how the proposed order of increments improves the time to delivery.

Intuitive explanation: Waterfall model versus Incremental model

It is obvious that dividing the problem into parts which can be developed in parallel is superior to serial development or developing the whole system as one entity in terms of time to delivery. The latter is assumed to be the main objective, hence the effort is not taken into consideration. The superiority of the incremental model is true as long as the division does not mean that the sum of the development times of the increments is much longer than the development time for the waterfall model or that the division into increments introduce much more faults than the waterfall approach. If the division into increments does not affect neither the development time nor the fault content then incremental development is superior due to the possibility of doing development and certification in parallel.

The question is in what cases the above is not true. To highlight this an example is introduced, but prior to the example a model describing how the reliability grows as faults are found must be formulated.

A simple failure occurrence model

A very simple failure model is to assume a certain number of execution cases (denoted N) of which a certain number will be executed with an erroneous result (denoted f). The faults lurking in the software (execution cases) are experienced as failures as they are executed. This simple model is illustrated in fig-

ure 1.

N = No faults + Faults (f)

Figure 1. A simple failure model.

A more advanced model, but similar model, has been used to simulate failure data to compare some different software reliability growth models (eg. Jones [14]). The model can be used to generate times between failures by random sampling with replacement or merely to calculate mean time between failures where either the failures can be corrected or not. The mean time between failures (MTBF) becomes:

$$MTBF = N / f \quad (1)$$

Equation 1 will be used to generate the time between failures to model the reliability growth of the software as faults are corrected.

Example

Introduction The example contains two parts, first a comparison between the waterfall model and the incremental model and then secondly an evaluation of the proposed order of increments compared to an arbitrary order. Before doing the comparisons, some assumptions and some definitions are needed.

The following general assumptions are made:

- The system can be divided into a number of equally large increments. This implies that the development of the increments take the same amount of time and also that each increment contains the same number of execution cases and therefore also the same number of faulty execution cases.
- The incremental development is assumed to be made in series, i.e. only one increment is developed at the time. The certification of an increment is, however, made in parallel with the development of a new increment. This assumptions is negative for the incremental model, since one of the major advantages with the incremental model is the possibility to develop increments in parallel.
- The times to perform the execution cases are assumed to be the same and it is also assumed that an execution case takes one time unit to execute. This time unit is not necessarily only the actual processor time, but some time to perform the case in general including user actions. This time unit is the same as the time referred to below when discussing the development and certification times respectively.

- Increments under certification, while others are developed, are tested as long as the development proceeds. The last certification is made until the reliability requirement for the system is fulfilled. The latter is also valid for the waterfall model when certifying the complete system prior to delivery.
- The example will not include any high risk increments, since it is difficult to compare a waterfall approach where the critical part is included as part of the system and hence not particular visible, with an incremental approach where the visibility of the critical parts is obvious through the division.

The definitions of execution cases (N), faulty execution cases (f) and MTBF are needed to model the reliability growth and the initial reliability as the software is brought to the certification team. This is, however, not enough, the following is needed as well:

- Development time for the waterfall model – X,
- The number of increments – n,
- Development time for an increment – X * c / n, where c is a scaling factor due to the division into increments. The factor can either be larger than one due to that the division into increments take some time or less than one due to that the development of an increment may go faster since it is a smaller piece of software.
- Reliability requirement in terms of MTBF – MTBF(req.),
- The number of faults in an increment – f * b, where b is a scaling factor describing that the division into increments may have changed the number of faults introduced in the software. b can either be larger than one or less than one with similar arguments as for the development time and the scaling factor c.
- Usage probability for increment i – p(i).

The scaling factors c and b can be determined, under the given assumptions and by letting one of the factors be constant while varying the other, so that the time to delivery with the waterfall model and the incremental model becomes the same (see below).

Waterfall model versus Incremental model The parameters are assigned figures, which simplifies the discussion considerably. The following figures will be used:

X = 1500, N = 1000, f = 20, MTBF(req.) = 200, n = 4, c = 1.2, b = 1.2 and p(1) = p(2) = p(3) = p(4) = 0.25.

The example is illustrated in figure 2.

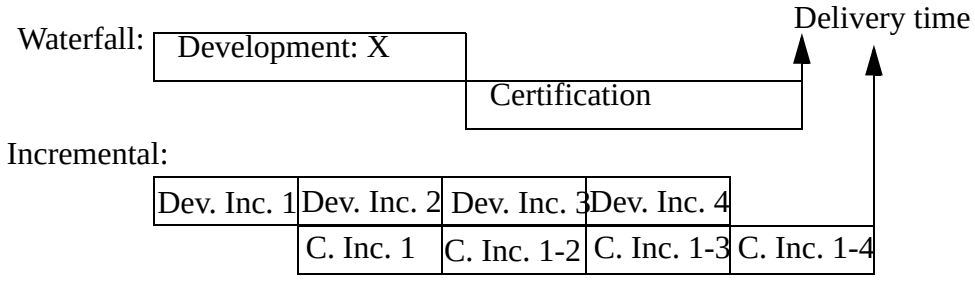

Figure 2. An illustration of the example.

Figure 2 only gives a schematic view of the problem hence the length of the different approaches shall not be judged from the figure, instead this is one of the important questions to answer in the comparison. The following questions are posed:

1. What is the time to delivery (TTD) for the waterfall model?
   TTD = 1500 + 1000/20 + 1000/19 + … + 1000/5 = 3014
   The last MTBF is equal to the requirement, i.e. MTBF(req.) = 200.
2. What is the time to delivery for the incremental model?
   The TTD consists of the development times for the four increments and the residual certification time to fulfil the reliability requirement. The certification time added depends on the certifications made in parallel with the development of increments 2, 3 and 4.
   MTBF values for the certification of increments during development of new increments are presented in table 1.

TABLE 1. MTBF values during incremental certification

	Inc. 1	**Inc. 1+2**	**Inc. 1+2+3**
MTBF(1)	**42**	**71**	**83**
MTBF(2)	**50**	**83**	**94**
MTBF(3)	**62**	**100**	**107**
MTBF(4)	**83**	**125**	**125**
MTBF(5)	**125**	**-**	**-**
Sum MTBF	**362**	**380**	**409**
Dev. time next inc.	**450**	**450**	**450**
MTBF after cert.	**250**	**167**	**150**

The figures in table 1 show that 5 faults are left in increments 1, 2 and 3 and increment 4 contains 6 faults, hence the certification means removing 7 faults to achieve the reliability requirement.
TTD = 4 * (1500 *1.2 / 4) + 1000/11 + 1000/10 + … 1000/5 = 2737 (<3014).

3. If b is 1.2, what value of c gives an equal time to delivery for the waterfall and the incremental model?
   The time to delivery (TTD) becomes as follows for some different values on c: c = 1.3 => TTD = 2887; c = 1.4 => TTD = 3037 and c = 1.5 => TTD = 3096.
   The breakpoint for c in this particular case is close to c = 1.4, (cf. 3014).
4. If c is 1.2, what value of b gives an equal time to delivery for the waterfall and the incremental model?
   The time to delivery (TTD) becomes as follows for some different values on b (or number of faults per increment): b = 1.4 (7 faults) => TTD = 2820; b = 1.6 (8 faults) => TTD = 3035 and b = 1.8 (9 faults) => TTD = 3097.
   The breakpoint for b in this particular case is close to b = 1.6, (cf. 3014).

The results in item 3 and 4 are non-linear and may vary unexpectedly, due to that the failure free execution at the end of the certification is not used. The time to delivery is more sensitive to a prolonged development time than a poorer reliability in the increments, which is shown by the values c and b in item 3 and 4.

It is clear from the example that the incremental model gives the same reliability in a shorter time period. Therefore the incremental model ought to be used to manage the quality of software. The model provides a better trade-off between reliability and time to delivery than the waterfall model. The incremental model would be even more superior if the increments were developed more or less in parallel and not in series as depicted here.

Incremental model: Arbitrary order versus optimum order In the previous part of the example, it was assumed that each increment was used with the same probability, i.e. equal values on all p(i). In such a case, the order of increments is irrelevant, but this equal distribution is not the normal case because for most applications there are some services or functions that are used more frequently than others. Therefore this case will be considered here, when illustrating the advantage with the proposed development order of increments. High risk increments are not considered as pointed out above.

Let p(1) = 0.50, p(2) = 0.30, p(3) = 0.15, p(4) = 0.05 and it is also assumed that the reliability requirement has been raised, due to a more demanding procurer, to 400.

The MTBF after certification for the entity under certification is known, but in reality not for the separate increments and in particular the MTBF at the start of certification is unknown. The latter is, however, not the case here where the objective is to illustrate the advantages and not to do an analysis of real data. After removing the last fault in an increment or in the system, it is assumed that the MTBF can be set to be 500 time units, which is twice the

value of the last MTBF for a particular increment and also higher than the requirement.

Arbitrary order of the increments, for example 3, 1, 4 and 2, is considered first. From the previous part of the example, it is known that the MTBF of the first certified increment becomes 250 (see table 1). The MTBF of the next increment when it is brought to certification is 42 (1000 / (4 * 6)) (here increment 1). This means that the initial MTBF for increment 1 + 3 becomes: 42 * 0.50 / 0.65 + 250 * 0.15 / 0.65 = 90, with 1 fault remaining in increment 3 and 6 faults in increment 1. The calculation of the MTBFs becomes a little more complicated since the testing is made according to the usage probabilities, i.e. in this case 0.15/0.65 and 0.50/0.65 which are the relative probabilities for these two increments. The MTBF values are shown in table 2.

TABLE 2. MTBF values for an arbitrary order of certification

	Inc. 3	**Inc. 3+1**	**Inc. 3+1+4**
MTBF(1)	**42**	**90**	**146**
MTBF(2)	**50**	**96**	**208**
MTBF(3)	**62**	**106**	**-**
MTBF(4)	**83**	**122**	**-**
MTBF(5)	**125**	**-**	**-**
Sum MTBF	**362**	**414**	**381**
Dev. time next inc.	**450**	**450**	**450**
MTBF after cert.	**250**	**154**	**208**

After certification of increments 1+3, 2 faults remain in increment 1 and 1 fault in increment 3, while after certification of increments 3+1+4, 1 fault remains in increment 3 and all 6 faults remain in increment 4. Therefore the number of faults brought to the certification of the complete system is 13 (1+6 +6), where 7 of them are removed, see table 3. This leaves 1 fault in increment 3 and 5 faults in increment 4. The certification was performed until the reliability requirement was fulfilled. The MTBF values together with the total certification time for the certification of the complete system is shown in table 3.

TABLE 3. MTBF values for the final certification period

MTBF	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Sum**
Arbitrary order	302	305	308	314	327	365	440	-	2361
Optimum order	171	172	174	205	215	340	346	421	2044

The same procedure is made for the optimum order, i.e. in this case the increments are taken in usage order. The total certification time and the MTBF values are also shown in table 3. The certification time becomes shorter with the proposed order of the increments. This certification leaves 2 faults in incre-

ment 3 and 6 faults in increment 4, while increment 1 and 2 are free from faults. One more fault is left in the software, but on the other hand the requirement is fulfilled which must be the main objective, i.e. the ‘right’ faults must be removed in as short time as possible.

Conclusions example Based on this example, it can be concluded that the incremental approach is better than the waterfall model and the proposed order of increments is superior to an arbitrary order, when it comes to an optimum relationship between reliability and delivery time.

CONCLUSIONS

Two important principles proposed in Cleanroom, incremental development and certification, have been discussed. The benefits with both incremental development and certification have been emphasized and in particular the advantages with combining incremental development with certification techniques have been presented. Incremental development and certification are methods which support managers in their work to stay in quality control in terms of reliability and software release time.

In realistic cases incremental development is superior to the waterfall approach and also to the spiral model since the latter does not allow for a step by step certification procedure. The superiority is due to the degrees of freedom with the incremental model and in particular the opportunity to perform work on parts to be delivered in parallel. This conclusions does, however, require that the two scaling factors introduced do not grow too large.

The proposed order of increments is superior to an arbitrary order of increments, hence providing a higher reliability in a shorter time period. The possibility to develop as high reliability software in as short time period as possible is a major challenge and thus incremental development ought to be applied. Some studies indicate that high reliability is highly correlated with high maintainability (eg. Stålhane [15]), which means that the result presented becomes even more important.

Managing software quality means adopting incremental development and certification, which provide higher reliability (and maintainability) in a shorter time period, hence allowing for on-time delivery with reliability control and fulfilment of the quality requirements.

REFERENCES

1. Mills, H. D., Dyer, M. and Linger, R. C. ‘Cleanroom Software Engineering’ *IEEE Software*, pp. 19-24, September 1987.
2. Mills, H. D., Linger, R. C. and Hevner, A. R. *Principles of Information Systems Analysis and Design* Academic Press Inc. 1986.
3. Linger, R. C., Mills, H. D. and Witt, B. I. *Structured Programming Theory and Practice* Addison-Wesley Publishing Company, 1979.
4. Cobb, R. H., and Mills, H. D. ‘Engineering Software Under Statistical Quality Control’ *IEEE Software*, pp. 44-54, November 1990.
5. Whittaker, J. A. and Poore, J. H. ‘Statistical Testing for Cleanroom Software Engineering’ *Proceedings 25th Annual Hawaii Int. Conf. on System Sciences*, pp. 428-436, Hawaii, USA, 1992.
6. Currit, P. A., Dyer, M., and Mills, H. D. ‘Certifying the Reliability of Software’ *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, pp. 3-11, January 1986.
7. Boehm, B. ‘A Spiral Model of Software Development and Enhancement’, *IEEE Computer*, pp. 61-72, May 1988.
8. Runeson, P. and Wohlin, C. ‘Usage Modelling: The Basis for Statistical Quality Control’, *Proceedings 10th Annual Software Reliability Symposium*, pp. 77-84, Denver, Colorado, USA, 1992.
9. Musa, J. D. ‘Operational Profiles in Software Reliability Engineering’ *IEEE Software*, pp. 14-32, March 1993.
10. Parnas, D.L., van Schouwen, A. J. and Kwan, S. P. ‘Evaluation of Safety-Critical Software’ *Communications of the ACM*, Vol. 33, No. 6, pp. 636-648, June 1990.
11. Musa, J. D., Iannino, A. and Okumoto, K. ‘*Software Reliability: Measurement, Prediction, Application*’ McGraw-Hill, New York, 1987.
12. Poore J. H., Mills H. D. and Mutchler, D. ‘Planning and Certifying Software System Reliability’ *IEEE Software*, pp. 88-99, January 1993.
13. Wohlin, C. and Runeson, P. ‘Certification of Software Components’, Submitted to *IEEE Transaction on Software Engineering*, 1993.
14. Jones, W. and Gregory, D. ‘Infinite Failure Models for a finite World: A Simulation Study for the Fault Discovery Process’, *Proceedings Int. Symposium on Software Reliability Engineering*, pp. 284-293, Denver, Colorado, USA, 1993.
15. Stålhane, T. and Wedde, K. J. ‘The Quest for Reliability – A Case Study’, *Proceedings 2nd Int. Conf. on Achieving Quality in Software*, pp. 309-320, Venice, Italy, 1993.

Biography: Claes Wohlin

Claes Wohlin received a Ph.D. from the Department of Communication Systems at Lund University, Lund, Sweden. He has five years of industrial experience from software projects. Currently he is associate professor at the department of Communication Systems, where he is responsible for education and research in the area of software engineering in telecommunications.

Title and Synopsis

Managing Software Quality through Incremental Development and Certification

A method for an optimum combination between on-time delivery and software reliability through incremental development and certification is presented.