SOFTWARE DEVELOPMENT: THE STAIRCASE APPROACH

K. Frühauf and K. J. Jeppesen

Brown, Boveri & Cie, Baden, Switzerland

Abstract. Modelling of the software development process for large software systems provides a means for controlling software projects. This paper presents a new model , the Staircase Approach, which takes into account the merits, applicability, and limits of the well-known Waterfall, Evolutionary Development, and Fast Prototyping models. In a nutshell, this approach utilises the ideas of Evolutionary Development for system level activities, the principles of the Waterfall Model for the lower level activities, and is suited extremely well for practising Fast Prototyping. Experiences gained in applying it indicate that the Staircase Approach is more suited than conventional models in achieving the objective of better software project control.

Keywords. Computer software; software engineering; software management; software quality assurance.

INTRODUCTION

Three dominating models, describing the software development, can be currently identified. The best known is the Software Life Cycle or Waterfall Model (e.g. Boehm, 1981). The work of Belady and Lehman (1976) on Evolutionary Software Development started back in the middle of the seventies. We are not aware of any reports, except Gilb (1983), about experiences applying their ideas. In recent years many software engineers can remember that Rapid or Fast Prototyping is an efficient way to check the feasibility of solutions. Prototyping can support or even replace the first phases in the Software Life Cycle, but it can not be used to describe the entire software development process. An example of fitting Fast Prototyping into the Software Life Cycle can be found in Matsumoto (1986).

In order to enable the readers to judge the applicability of our approach in their environment, we list here preconditions which have had a major impact on the selection of our approach:

- software is embedded

  The software product is part of the delivery comprising computer and telemetry equipment.

- software product is resold ten to a hundred times

  The software product - a system of the BECOS family (Goudie, Davis, and Spatz 1984) - is resold, but in relatively small quantities compared with e.g. compilers. It must, however, be adapted every time to the needs of the actual customer. Thus it is neither a single shot "customer tailored" nor a mass produced (by copying) "off the shelf" product.

- data are always customer specific

  Even if the algorithms could be reused without change, the data - which is considerable for a power system - are by nature customer specific. The test on the actual hardware configuration and with the actual customer data requires a certain amount of time: the freeze of the software and a controlled transfer from the product development to the project is a necessity.

- delivery times last from 6 to 36 months

  The delivery time depends on

the size of the system. The product release, on which the delivery is based, must be known around 6 to 12 months ahead, at the tendering stage. With the biggest systems, customisation is carried out on more than one release. This puts some constraints on the characteristics of the consecutive releases, e.g. upwards compatibility.

- number of products have already been developed

  Feasible patterns for solutions are available. The development of a new product can start with reusing big lumps of available software.

- requirements are unstable

  For products of a certain size and complexity, neither the supplier nor the customer completely understand the problems of the particular application and, therefore, cannot specify exactly all requirements. The long project lifetimes - from the customer's initial idea until final acceptance of the system can last as long as ten years - accentuate this problem even more.

Developers want to describe the development process in terms of its different phases and tasks. Management wants to establish a strategy for product development and delivery projects, i.e. taking the entire system and its total lifetime into account. It is difficult to achieve these two aims with a single model.

In the following section we discuss how the Waterfall Model can be used to describe software development. The basic principles of the Waterfall Model - phases and milestones - turn out to be a kind of "natural law". We will also demonstrate, however, that the model is inadequate for establishing a development strategy on a system level. The list of objections to the Waterfall Model provides the justification for the Staircase Approach.

Staircase Approach takes into account basic principles of the Waterfall Model, but avoids its system level inapplicability by using ideas from the Evolutionary Software Development approach. System level is that level of abstraction, where the internal technical details are of no relevance. The main objective is to enable management to control large software developments.

WATERFALL MODEL : MERITS AND DEFICIENCIES

As the Waterfall Model is the only theory which is not applicable "as is" in our environment, we will consider its advantages and drawbacks in the following section.

In the Waterfall Model the software development process

- is sequential

  The development process is subdivided into logically sequential phases. They reflect the natural sequence of tasks (e.g. specification, design, coding, test) to produce an item of software. This logical sequence is, in many cases, mapped on a time scale so that it leads to a strict, chronological sequence of these tasks.

- is iterative

  Mistakes detected in later phases reinitiate the process at some earlier phase. This iteration attempts to cover the fact that a task, e.g. design, is not finished "for ever" at a certain milestone. The consequence is that all phases are finished only when the last milestone is reached.

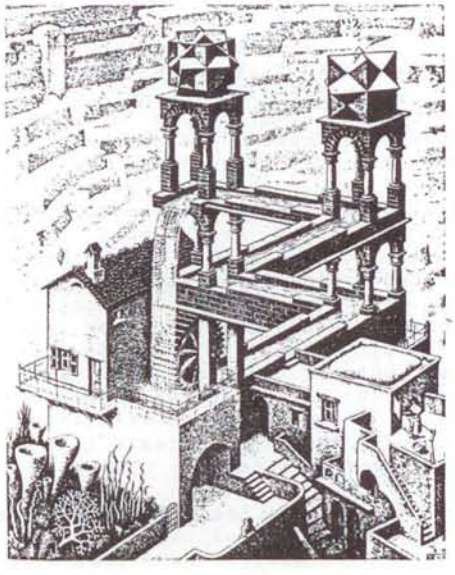

Fig. 1 : Waterfall, by M.C. Escher (lithograph, 1961).

This contradiction - strictly sequential phases vs. iterations - is one of the reasons for the difficulties regarding software projects. Another reason is the confusion caused by the often neglected fact that the objects of the tasks in the different phases are not identical (e.g. specification of the **system**, design of **subsystems**, coding and testing of different **modules**).

While we can identify a single task, for example, "system specification", we have to cope with a number of "subsystem design" and even a larger number of "module coding" tasks. To force, e.g. that the coding of all modules be in one time slot (phase) of a large project, would be from a managerial (e.g. resource demand) and technical point of view (e.g. kernel facilities needed first to ease test of applications), often foolish.

Therefore, we have refined the interpretation of a phase in the Waterfall Model. Example of our interpretation for, e.g. coding: "A module can only be coded when the enclosing subsystem is designed but irrespective of whether the other subsystems are designed or not" instead of the usual "all design must be done before coding". For this interpretation, we use the term activity, e.g. module coding activity. By this interpretaion we are free to assign these **activities** to the adequate time slots in the project, i.e. to our **phases**.

The phases thus serve a well defined **managerial** purpose. Strictly sequential phases, delimited by milestones, are a tool for management in budget allocation and progress control. To enable a definitive closing of phases, activities on different elements of the software structure must be each assigned uniquely to a phase: when all activities assigned to a particular phase are finished, the phase itself is finished. Thus iteration is replaced by subdivision of tasks into "try it", "do it", and "modify it" types of activities and by the assignment of these to - possibly different - phases.

Although the Waterfall Model is not suited for the overall project, it is perfectly adequate for the development of a single function or block of related functions. It describes all necessary activities for development of a software item by a limited number of people. Lower level management has the means to:

- control the progress of the individual groups since the status of activities is visible,
- assign the different activities (e.g. specification, design, coding) in certain areas (e.g. human-computer interaction, telemetry) to different groups in the organisation based on their skills and resource availability,
- continuously improve effort estimates as every current activity completely defines the following activity, including estimates.

As soon as the work-breakdown results in elements requiring less than three person-years effort, the Waterfall Model is the way to run the development.

Our concern is for software products of a much bigger size and thus of a higher complexity, where activities must be carried out by different groups. Here, the Waterfall Model does not adequately describe the development process. Its application to large projects could be even dangerous. One of the dangers is that large amounts of effort might be put into the first phases with no feedback on usability. Additionally, the tendency for "overperfecting", e.g. the design, without real progress on the project, could result. Specification is another example: engineers can always argue that having an unsound specification bears a very high risk for the project. The manager knows that and has therefore no weapons to fight this argument. Project termination could remain as the only way to escape from the endless loop of refining the specification.

Another danger of the stringent application of the Waterfall Model to the overall project is the frustration of the developers. It might be very demotivating to see the product for years only on paper, but not "alive". The developers could start to doubt whether it will ever work.

THE STAIRCASE APPROACH

Definitions

The above discussion indicated that we have specific interpretation for some widely used terms. We summarise here our definition of the terms necessary for the understanding of the further sections of the paper.

Project. The planned pattern of all actions necessary to produce a software system.

Phase. A period of time within which certain results - a defined milestone - must be achieved. The chosen phase name is, usually, the name of the dominant activity in that period. Note the phase definition is in terms of time.

Milestone. Point in time defined by the existence of the predefined set of software items, including their approval.

Activity. Work unit carried out on an element of the system structure at a certain level, e.g. system specification, subsystem design, module coding. Note that this definition is actually a type definition: module A coding and module B coding are different activities of the same type (module coding). An activity must be uniquely assigned to a phase.

Product. The result of a software development project, having a specific set of identified constituents, and having, at each stage of development, a defined functionality.

Release. The software items constitute a predefined functionality of the product being developed.

Variant. A variant is the basis for operational software. It is a subset of a release tailored to the needs of a particular user and supplemented by the static data of the process to be controlled. A variant may include software items developed for customer specific requirements.

Objectives

The objectives for the Staircase Approach were:

- Define the development in stages so that the product can be, at every stage, described and marketed as a complete product.
- Provide a framework for the system level activities in addition to the low-level activities according to the Waterfall Model.
- Enable a bringing up an initial version of the system as quickly as possible in order to:
  - prove the feasibility of the idea,
  - demonstrate the system to prospective customers,
  - expose the system to the market for early feedback on the commercial value of the product,
  - provide motivation to developers.
- Establish regularly running versions of the system to
  - allow management to control development on a system level,
  - make sure that the system is in a consistent state,
  - allow quality and functionality to be regularly verified.

Elements of the Staircase Approach

The Staircase Approach comprises three main elements (see Fig. 2):

1. Release concept

   Step-by-step enhancement of the software product reaching, at regular intervals "another floor with doors to the prospective customers", i.e. releases as basis for variants - hence the name "Staircase Approach".

2. Release development

   Four phases with assigned activities and defined milestones. The schedule is dictated by the release period.

3. Customisation in variants

   Six phases with defined milestones requiring customer approval. Product tailoring to the customer requirements does not allow as high a standardisation of activity assignments as does release development.

Release Development

The heart of the Staircase Approach is the development of the system in a number of regularly issued releases. A release is defined by a set of functional and quality requirements. The frequency of releases depends on the size of the product and on the number of persons involved in the development. For power system control applications the period shall be between four and twelve months. The development of every release can be considered as a separate project composed of the following phases:

- Release Planning

  Specification of requirements, overall design, cost estimation, personpower planning, and feasibility studies (including fast prototyping) are the main activities in this phase. Sales of variants based on a given release will be launched only after this phase is finished. The milestone is

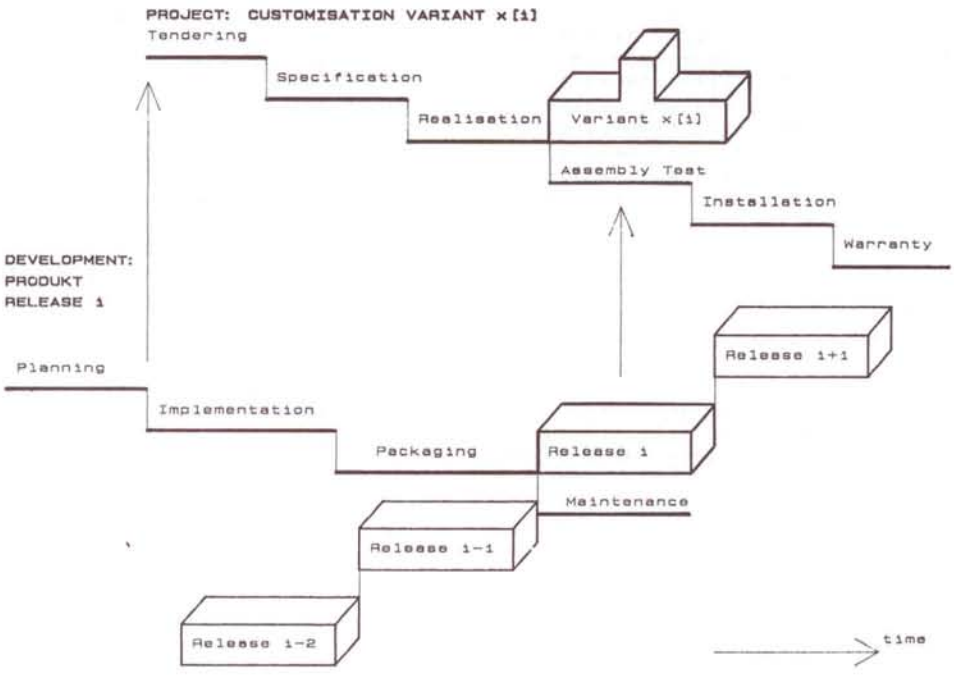

Fig. 2 : The Staircase Approach.

"sales launching" and is achieved when the entire system level documentation of the release is available and approved.

- Release Implementation

The main activities are the detailed specification of functions and design of the modifications to the current release (including reviews), test design, coding and test of the modifications, integration of the modifications and system test.

The main difficulty in this phase is the work break-down. More than one function can require a modification to a particular module. The work must thus be sequenced either function-by- function (so that at no point in time do two developers have to modify the same module) or module-by-module (so that all modifications of a particular module for all the functions are done as a work unit). The former is much easier to manage but the danger of "slipping into corrupted design" must be recognised and avoided.

- Release Packaging

Configuration of the release, formal acceptance test, project audit, and product audit are the dominant activities in this phase. The configuration provides a comprehensive list of items forming the release (comprehensive means that data and documents are included). The formal acceptance test, carried out by an independent group, confirms the consistency of the items and the performance according to the specified requirements. The project and product audit provide additional confidence for the quality of the development process and of the product.

- Release Maintenance

One of the merits of the approach is that this phase comprises only corrective maintenance. All required enhancements must be considered in release planning. Providing that development is directed toward upwards compatibility and that site updates are enforced, the release maintenance can be abandoned as soon as the next release is available. This requires some of the errors to be corrected twice: First in the release it was reported for and additionally in the subsequent release.

The concern of all activities in the release planning and packaging phases is the entire system. Only a small team is involved at these stages.

During release implementation, the functions are the main concern. Their implementation can strictly follow the "natural law" which is the middle part of the Waterfall Model: Detailed Specification, Detailed Design, Coding and Unit Test, Integration. The size of the team depends on the functionality increment and can vary from release to release. Many developers or even groups of developers can implement functions in parallel, each working strictly according to the "natural law".

Release maintenance is concerned with the errors reported by the users of the release. The entire system will be brought to a consistent state in the form of release upgrade with some of the errors removed. Between two releases up to four upgrades could be provided for the users.

Customisation in Variants

Product development is directed by the market or, more precisely, by our perception of what the market requires and what the technology provides. In the delivery projects, i.e. projects resulting in a variant, the target to meet is the satisfaction of a particular customer.

The variant is derived from a specific product release. The customisation is basically the specification of the customer's requirements and the configuration of the corresponding release subset supplemented by the customer specific data. Most of the customers, at least in our application area, will have requirements not considered in the planned product releases. These customer specifics will be implemented within the framework of the delivery project and, unfortunately for the customers, will be fully charged for. The advantage of having standard requirements (of a given product release) is that their development costs are shared by all buyers of the product.

The delivery projects are chronologically subdivided into the following phases: tendering, system specification, realisation, assembly test, installation, warranty, and maintenance. The following sections describe which activities are carried out in these phases, with special emphasis on what is available with each particular milestone. Note that the concern of all activities is the variant.

- Tendering

  The most important activity is to find the best matching product release and to identify the deviations between the expectations of the prospective customer and the requirement specifications of the chosen release. These deviations are the main topics of the contractual negotiations. The milestone delimiting this phase is the contract signature by both the supplier and customer.

- System Specification

  All uncertainties between the supplier and customer which the contract negotiations did not take into account must be settled in this phase. The contract must be reviewed and the requirement specifications refined to the necessary level of detail. That level is reached when the hardware items can be ordered and the variant can be tailored. The customer shares here the work and responsibility: He must provide all data - correctly and on time - describing his application process.

  Providing verification procedures for factory acceptance tests at this stage and their approval by the customer will help a great deal in clarifying requirements. Because the verification procedures are the most user-friendly description of the product, their inclusion in the contract is actually our goal. However, the customers are not ready yet to accept this idea.

  The milestone of this phase is the approval of the detailed specification (and, hopefully, of the verification procedures) by the customer.

- Realisation

  The product release is tailored and the variant built and loaded with all data needed to model the process to be controlled. If customer specifics are required, it is during this phase in which the required functions are implemented, i.e. the variant is supplemented by the customer specific software items. The development of these functions follows exactly the same path as for release implementation. The system test is accomplished according to the verification procedures prepared for the factory acceptance test. The tests should include the user manuals. The quality assurance engineer will certify the achievement of the milestone "system test passed".

- Assembly test

  The independently tested software system is brought to the independently tested actual hardware. The purpose of the phase is to test, prior delivery, the hardware-software combination in supplier's test

field. Corrective maintenance will be made. Latest by the start of the factory acceptance test all user manuals must be ready. The succesful completion of the factory acceptance test will be certified by the customer and the variant version archived. The approval of the verification procedures for the customer site acceptance tests at this stage is highly recommended.

- Installation

  The hardware and software are installed and tested at the customer site. Software problems are either reported back home or corrected on site. The valid version of the variant is always the one on the customer site. The milestone is the successful site acceptance test certified by the customer. The software engineer must not forget to take home the last version for archiving. This is a prerequisite for any chance to reproduce and correct errors reported by the end user.

- Warranty

  Corrective maintenance based on problem reports from end users. The end of the phase is certified by the customer with the certificate of final acceptance. The complete version of the variant will be archived as well as the project folder containing the history of the project. A nice project leader will even write a project closing report with some figures and remarks useful for his or her colleagues.

- Maintenance

  Maintenance as generally understood. Bigger enhancements will be carried out as a new project or, if we are lucky, the next product release will do it. The luck is dependent on the compatibility of the customer specifics in the variant with the new release. Every maintenance lump is finished with the archiving of the new version of the variant and of the updated project folder.

EXPERIENCE GAINED

The main advantage in applying the Staircase Approach is the frequency of "deliveries" with stringent control of quality, value, cost, and schedule. The choice of the release period is a trade-off between the cost of release packaging and the prospective benefit of risk reduction. It is mainly dependent on the product size. The best approach is to limit the fund for a release and insist on provision of a release specification which can be implemented with that money.

The time between initial tendering and order is, in our business, roughly twelve months. Assuming a release period of six months, management must always have at hand plans for at least two releases ahead.

The application of the Staircase Approach to a two hundred person-years development effort of a product with the life-time of ten years results in ten to twenty small and thus manageable projects. With a release period of six months, they will last - from start of the release planning until the end of release maintenance - at most two years and consume ten to twenty person-years development effort. Note that the main cause of the two years time interval is the long tendering phase. The two phases of realisation and packaging together last the release period (e.g. six months); the maintenance phase lasts an equal period of time. Half of the release lifetime is used for the planning phase - enough room to extensively practise Fast Prototyping.

Fast Prototyping is used for three purposes. First, it is used for feasibility studies while specifying releases. The simulation of interfaces to existing software packages is another purpose. Before deciding to incorporate an application software package, the interface is prototyped. In addition to the feasibility demonstration, the known prototyping effort enables a more accurate cost estimate for the actual incorporation. Finally, the early presentability of the future product "surface" provides a higher confidence level by management as well as by the prospective customers. Both management and prospective customers can see what they will get for their money.

The release implementation, packaging, and maintenance phases are at any time in progress for a single release. Necessitated by its length, release planning of two consecutive releases will overlap. This enables high flexibility in allocating requirements to the releases depending on the current set of variant tenders and orders. A strong discipline in documenting and communicating the decisions within the team is a necessity, otherwise confusion about the final appearance of the concerned releases is the unavoidable consequence.

We encountered an interesting problem when applying this approach. The developers focus only on the next release. They are naturally concerned that funding may dry up, and thus there

will be no money for climbing the next steps on the staircase. They have difficulty recognising that the system development is not according to the Waterfall Model (which appears to be a more finite process). The consequence is that they tend to put the full functionality into the next release, although only part of the functionality was specified for. Providing the deadlines are met and the costs does not exceed the budget, this can lead to more-than-mature releases. The manager must, thus, carefully untangle the reasons of deadline slippage.

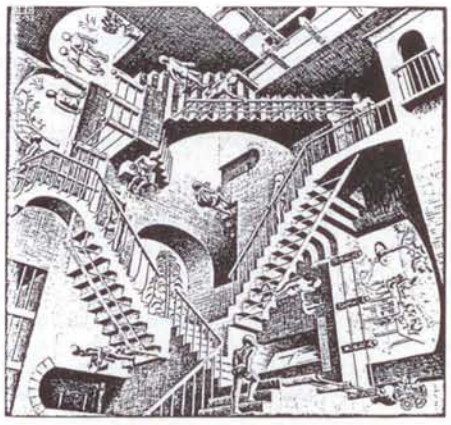

Fig. 3 : Relativity, by M.C. Escher (lithograh, 1953).

With the product releases planned sufficiently ahead it is much easier to negotiate with a prospective customer: only the requirements not met by the corresponding release must be negotiated. The delivery time of the best-matching release could be too late for the purchaser. Assuming upwards compatibility of the releases, this problem can be resolved by phased delivery: the customer will obtain successive releases until his ultimate requirements are satisfied. The crucial matter is to have a well-designed product so that upwards compatibility can be guaranteed. Everyone who has been told by an operating system supplier that the releases of his operating system are completely upwards compatible will soon realise that for process control applications this is a very difficult task !

Delivery time for a variant can be shortened significantly. If the variant does not contain any customer specifics the delivery time is dictated by the hardware delivery. Buying release functionality thus provides the customer with the advantage of lower price and shorter delivery. An additional advantage is provided by having a larger number of end users: all of them will contribute to error discovery. Remaining errors in customer specifics can, by nature, occur only on one site. In summary, the customer has less risk in purchasing a standard product release.

In Paige (1985), a distinction is made between software development for novel and sustaining products. The application of the Staircase Approach for development of sustaining products is straightforward. They have stable requirements so it is easy to reuse available solutions: the step to the first release, which should already be of value to end users, can be made within the expected time frame.

It is much more difficult to apply the approach for the development of novel products. The critical matter is the size of the product. The system architect's challenge is to identify the kernel part of the system which then becomes release one. This first release will usually take more than the intended release period, but it should not take longer than two periods. The second release will most likely be the first release of any value to an end user.

In both cases, the consistent application of the Staircase Approach encourages the reuse of software. With strictly limited funds allocated and the request for releases usable by end users the "must be invented here" syndrome can be very effectively fought.

CONCLUSIONS

The Staircase Approach is the synthesis of the well-known software development models. It is an attempt to use their merits and avoid their drawbacks. The evolution of a product is forced into regular, timely steps, and the developers are forced to let the product "out of their hands" for customisation. This frequent exposure of their results to other internal groups and to customers stimulates "craftsman pride" of the developers. The successful passing of all acceptance tests and audits provides job satisfaction and continued motivation.

But it is not easy to convince developers that every six months a complete product must be turned out without all their brilliant ideas being implemented. It is even more difficult to convince the salesmen that selling variants strictly adhering to the release functionality will create customer satisfaction and tip the balance sheet in their favour.

Customers in the area of power system control will soon accept this new situation. They will observe happy neighbours operating reliable systems and having forgotten all their extra wishes - partly because the next release will include their wishes for the price of a software update contract, and partly because they have learned, by using the system, that the benefit could

never outweigh the cost of their extras.

The prerequisite for the approach is a system architecture designed for easy extensions and a provision of all essential functions in the initial release. Thus, knowledge of the market and software engineering know-how are necessary.

ACKNOWLEDGMENT

The authors would like to express their appreciation to F. Merola for improving the language of the paper.

REFERENCES

Belady L.A., Lehman, M.M. (1976). A Model of Large Program Development. IBM Systems Journal, No. 3, 1976, pp. 225-252.

Boehm B.W. (1981). Software Engineering Economics. Prentice Hall, 1981.

Frđhauf K., and Sandmayr H. (1983). Quality of the Software Development Process. IFAC Safecomp 83, Cambridge University Press 1983, pp. 145-152.

Gilb T. (1983). Management Technoscopes. Unpublished book manuscript, 1983.

Locher J.L. ed. (1971).De werelden van M. C. Escher. Meulenhoff International Amsterdam, 1971. Source for photocopies of the lithographs by M.C. Escher.

Matsumoto Y. (1986). Requirements Engineering and Software Development : A Study Toward Another Life Cycle Model Proceedings of the Symposium Computer Systems for Process Control. Plenum Publishing Corporation, 1986.

Paige M.P. (1985). On Software Quality Assurance Strategy. Conference Record Eighteenth Asilomar Conference on Circuits, Systems and Computers, IEEE Computer Society, 1985, pp. 257-261.