THE INTEGRATION AND DISTRIBUTION PHASE IN THE SOFTWARE LIFE CYCLE

G. CASAGLIA - F. PISANI

OLIVETTI - DIDAU/DSM

VIA JERVIS, 77 - 10015 IVREA - ITALY

ABSTRACT

The software production process may be seen as three main phases:
definition and design, implentation and distribution. It is obvious that in a industrial environment the phases must have a comparable throughput.
In past years at Olivetti, design and implementation has been significantly increased, introducing a new set of programming tools such as:

- UNIX os plus a number of related tools (make, berkleynet, mail,....)
- Pascal+: an enhanced version of Pascal including monitors, as system programming language.

Special care has also been given to the final part of production process where all software components are integrated, finally tested and distributed to subsidiares and then to customers.
A number of management procedures and automated tools have been defined with the purpose of enhancing such integration/distribution process; among these, worth of note are the integration plan, describing the process managed by an integration control board, and release committee.
Two level distribution data base, system test and amendment data base are some tools supporting the process.
The presentation will sketch the whole software life cycle and then will concentrate in the description of the integration-distribution process. According to our experience this step may introduce a significant bottleneck. Removing such bottleneck can significantly increase the performance (and quality) of the entire process.
A detailed analysis of critical points and problems to be solved is derived, following our experience in developing an entirely new operating system.

1. INTRODUCTION

In 1980, starting a completely new software project, it was decided to introduce a new software life cicle and a completely new set of tools for software production. At that time software implementation was based on a number of different tools, depending on different projects, but a limited number of scenarios can be described:

i) small projects using target machine as support, and assembly languages plus various types of debugging aids.

ii) medium size projects accessing IBM T.S. system, using cross-tools and various means for transferring object code from cross-system to target systems (i.e. down- line loading, transport via compatibile media and so on). Assembler and, in some cases high-level languages, were used in system software implementation.

iii) medium size projects using IBM RJE facilities, plus cross-tools and assembly

languages.
Each project adopted a specific-implicit-life cycle model.

Due to the characteristics of the new project, large system software for a new line of minicomputers with a significant number of successive releases to be produced, it was decided to define a uniform software production environment, suitable for all development groups. A plan to have all groups migrating from their "private" environment to the new "software factory" was also defined.

The key points on which the new "software factory" was based are the following:

- use of minicomputers as development systems (PDP-11/70 and VAX), in order to be able to dedicate computing power to each project group and to be able to expand such capacity, according to the specific needs of such project groups.

- use of UNIX as operating system and the related development tools, as the most advanced development environment.

- distribution of a large number of terminals to the different project groups.

- use of Pascal as high level development language.

- network interconnection of all development systems in order to have the different groups exchange mail, documentation, source modules.

- connection target systems to development systems for down-line loading of programs.

- definition of a suitable global software life cicle.

2. PRESENT SITUATION

Today the environment is completed and successfully operational.
Just to give some idea, the present situation is as follows:

- About 30 development systems (PDP-11/70, VAX) are installed in seven different locations: four in Italy and three in the USA.

- A network is connecting all of them using dedicated, switched lines and satellite links.

- About 900 terminals are connected to the development systems serving about 1100 people involved in planning, implementation, QC, Sw distribution. This gives 1 terminal per 1.2 person.

- 90% of all the software produced is written in Pascal.

Fig. 2.1 is showing the general topolog of the network, being this the powerfull base supporting the development tools and metodology.

In the first two years of the project much care was dedicated to the environment and tools affecting productivity in the implementation phase (i.e. implementation language, debugging tools, computing equipments).
While more can be done in this area, that seems to be the area on which research efforts are concentrated, the two following years of our project have shown a large impact of the integration/distribution phase on global productivity.
The following sections are firstly dedicated to analize the global production process we have adopted and then to discuss some implications and then to consider possible evolutions or alternatives.

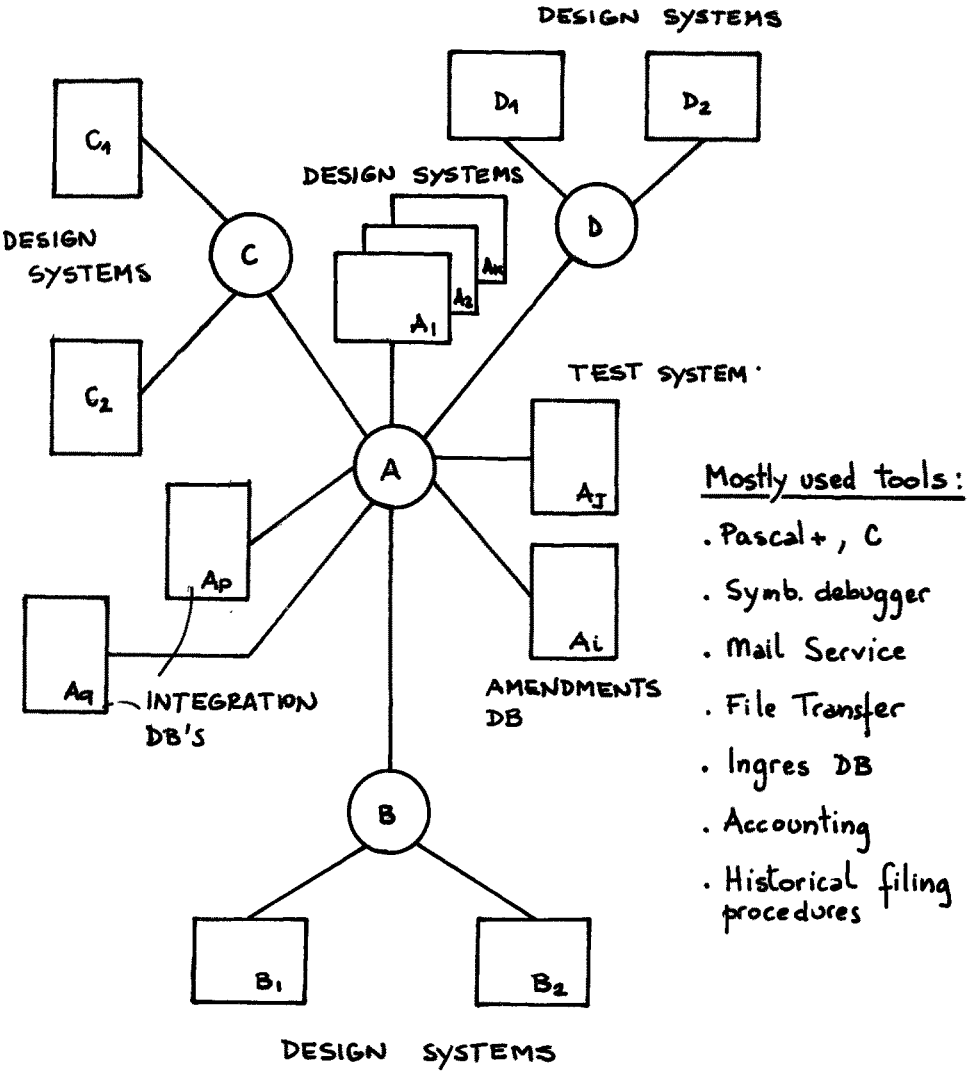

Fig. 2.1 Development Network Topology

3. ORGANISATIONAL ISSUES AND FLOW OF THE DEVELOPMENT PROCESS

Groups dedicated to the implementation of the software system, were organised according to a functional structure, like the one sketched in fig. 3.1

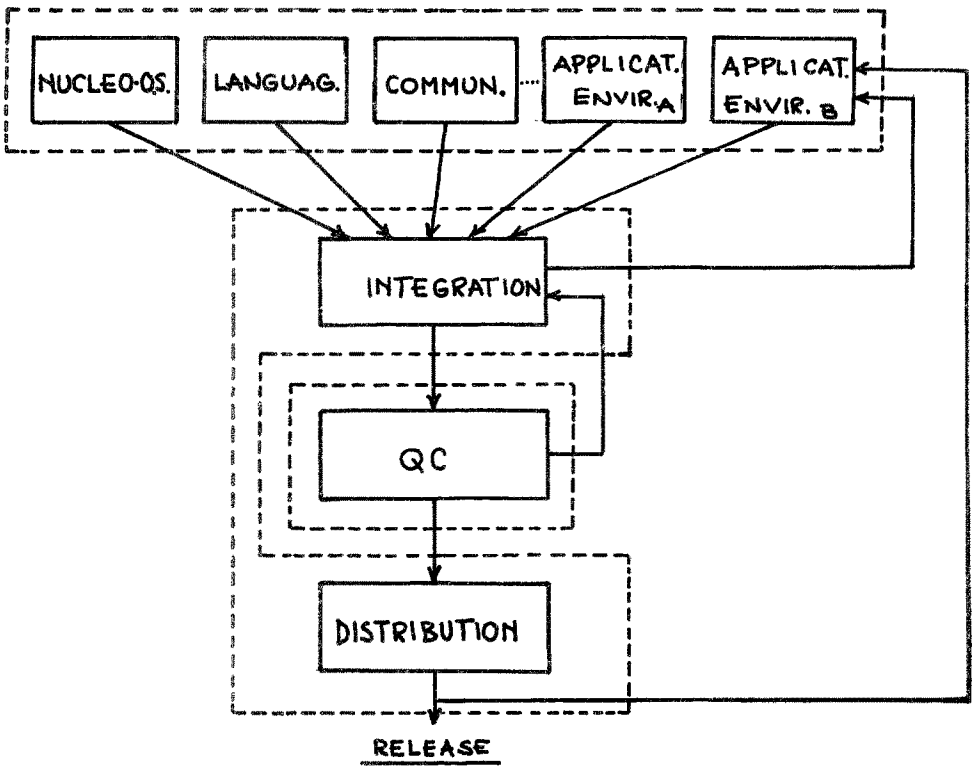

Fig. 3.1. Functional structure organisation

The main characteristics of such structure are:

- Implementation of all software parts is done in parallel up to completion of all components according to the functional specs, practically without feedbacks from Integration and CQ phases before the end of implementation phase.

- Quality control is performed in a large single external organisation independent from the implementation groups.

- Integration control and distribution is performed by a third independent group.

- The production process is practically divided in two parts:
  - Implentation
  - CQ, Integration/Distribution

- The process can be described as follows:
  . implementation of different components is completed independently
  . components are then Quality Controlled
  . they are then funneled to the integration process, integrated to form the complete system and then quality controlled, according to the various system-configuration.
  . At this point a complex feedback process among project, QC, integration-/distribution is started, in order to produce the various product versions.

In past years these structure has proved to be efficient for:
- implementing a large integrated systems
- efficient and fast implementation of a large quantity of different packages to be integrated in a unique systems
- organizing a strong control in the quality-control integration/distribution phases while the implementation phase is technology driven for a large period
- performing configuration control in the final phase of the release
- producing and managing a release version at a time.

4. PRESENT SOFTWARE LIFE CYCLE.

Software life cycle we adopted can be described using the cascade model (BOE 81) modified in order to reflect the independence between sw modules implementation process and product release production process.
Life cycle shown in fig. 4.1, has the following characteristics:

. Functionalities of sw components to be developed are defined during requirements analysis phase (REQA) driven by technological issues, rather than market requirements.

. Development of software components is following a rather standard cycle, where components implementation and QC test definition and implementation are proceeding in parallel.
Implementation of all components is proceeding independently.

. Market requirement are introduced later in the development process, in order to define the set of components to be included in a product release.
Such definition, plus the components definition, allows the definition of the actual Product Release Contents (PREQA), and the definition of Integration and Test Strategy (Integration Tree, Test Plan, ect.)
It is in this phase that additional developments are defined to complete Product Release Functionalities.

. Integration and test phase include high interaction among development groups. Distribution kit is prepared and tested during this phase.

. Integration phase ends with the actual delivery of the release to a System Test phase, validating functional contents through home tests (application environment emulation) and through beta test (pilot user environment test).

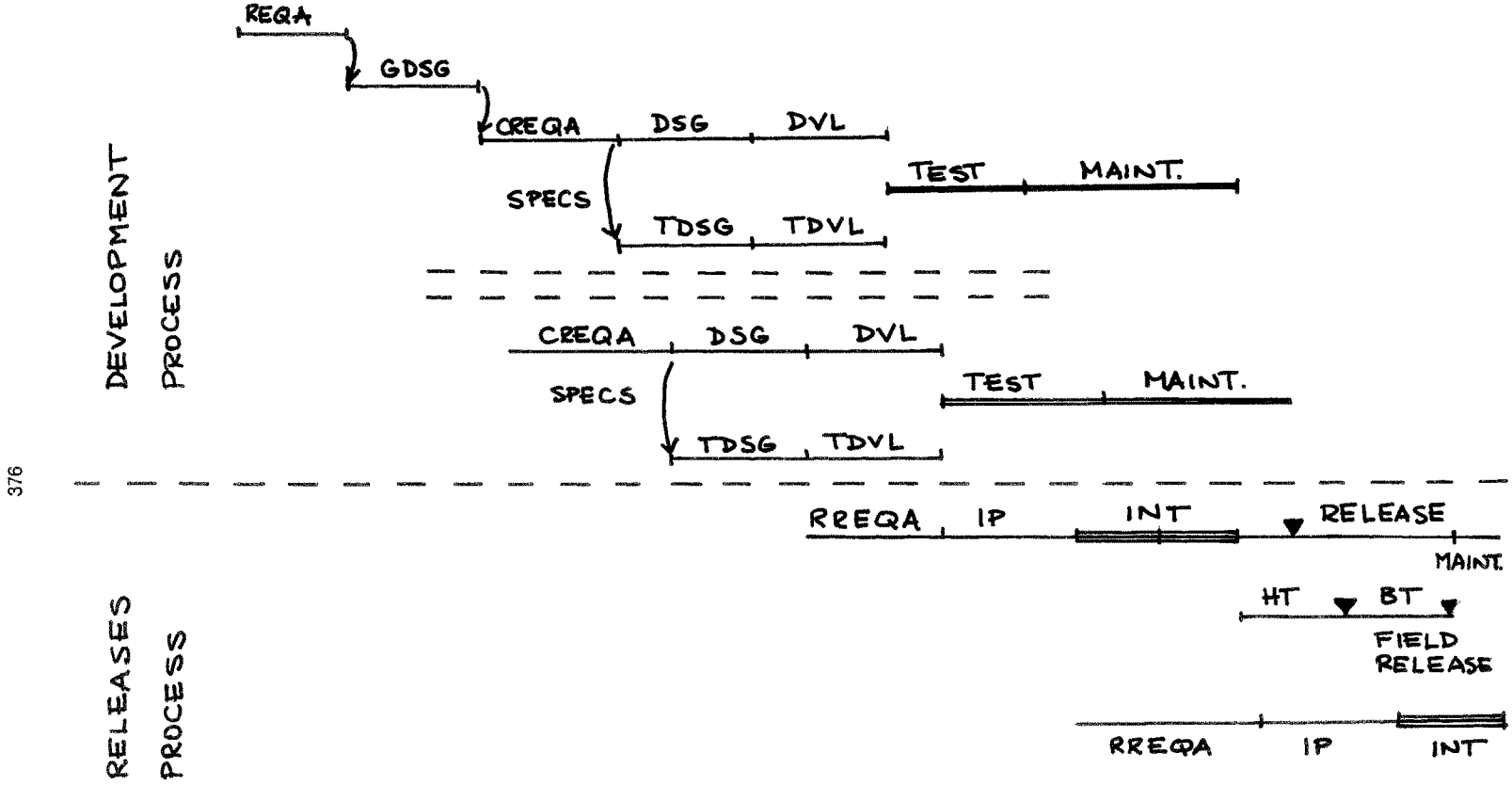

Fig. 4.1 Software Life Cycle

A complete life cycle include three-four time integration phase iteration for any components development phase.

5. INTEGRATION PROCESS

The Integration is an ordered process of building-up and testing of a set of software components of increasing complexity (fig. 5.1).

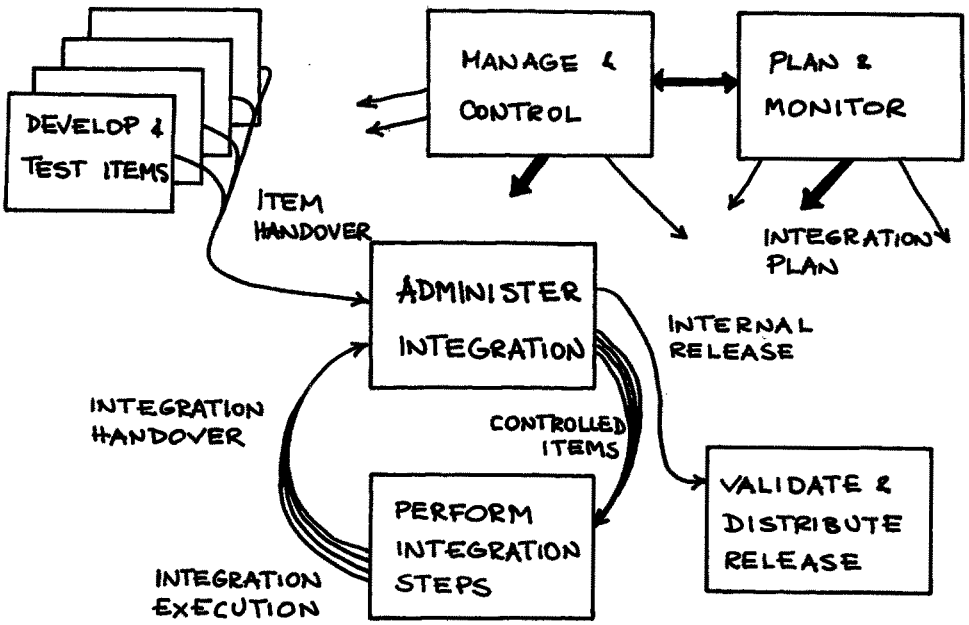

Fig. 5.1 The Integration Process

The integration process is highly variable and iterative, therefore it must be strictly controlled. (Table 5.1)

- Integration Programming
  - Includes technical planning, monitoring replanning
- Integration Execution
  - Construction and test
- Integration Administration
  - Includes Error handling
- Integration Control
  - Management & Change Change Control

Tab. 5.1 Integration Activities

Control of the process can be obtained having a high visibility of the process; such visibility is obtained having the software components flowing through three different and independent organisations:

. Development groups seen as producer of the components and the related documentation.

. Integration, in charge of receiving, managing and controlling the software library and organizing the documentation needed to build the product release.

. Quality Insurance, controlling the quality of the software received from Integration.

Fig. 5.2 shows the information flow among the different organisations.
An integration control board coordinate change control and conflict resolution activities, in order to have a smoothly converging process.

A typical Integration cycle is shown in fig 5.3, where critical mile stones are:

. delivery to integration of the last relevant software component;
. execution of a complete test phase, accepting software changes for errors during the test phase;
. non-regression test on final sw version, accepting controlled and authorized software changes;
. production of distribution kit.

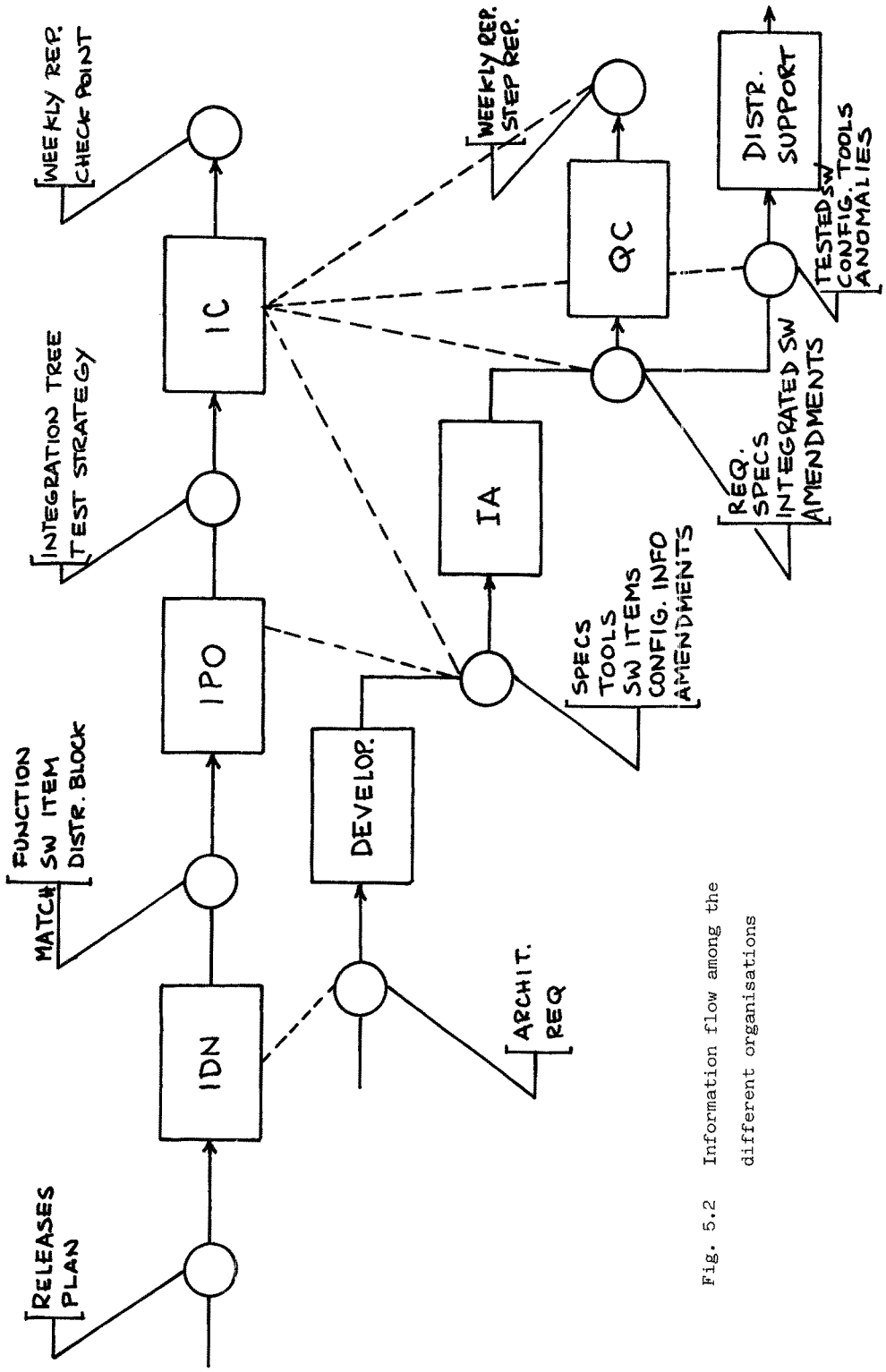

Fig. 5.2 Information flow among the different organisations

repeat {
ITEM HANDOVER
ITEM ACCEPTED
CONTROLLED SW HANDOUT
}
FIRST TEST CYCLE
repeat {
ITEM REVISION HANDOVER
ITEM REVISION ACCEPTED
CONTROLLED SW HANDOUT
}
FIRST DISTRIBUTION KIT
NO REGRESSION TEST
repeat {
PATCH HANDOVER
PATCH ACCEPTED
CONTROLLED SW HANDOUT
}
LAST DISTRIBUTION KIT
RELEASE

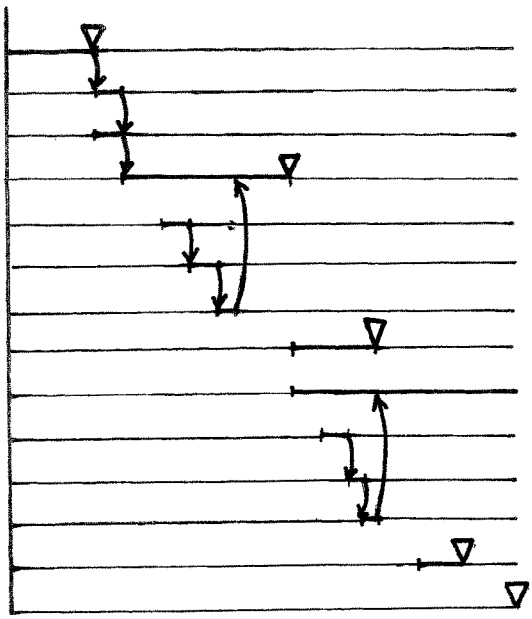

Fig. 5.3 Integration Life Cycle

6. INTERMEDIATE STATUS.

The environment described has shown the following caracteristics:

. Provides an efficient implementation process

. Implementation process, not integrated with the functional specs definition and plan definition, gives too loose connection and feedbacks between this two phases which are proceeding practically in parallel, up to the integration phase.

. Strong QC and Integration phases at the end of the process; this gives:
  + capacity of producing large complex configurations; but
  + long feedback-time among definition, implementation and integration
  . potential boottleneck in the final phase, as soon as the first versions are available and a significant number of different releases has to be managed.

7. EVOLUTION OF PRODUCTION PROCESS AND SOFTWARE LIFE CYCLE.

As the first release of the new product was distributed it became apparent a new problem, we can try to summurize as follows:

Present structure is targeted to manage a very long implementation period of a sophisticated technology driven software system; i.e. implementation period has

weak feedback from the market and is performed with large authonomy of project and programming people.
As soon as the sw product is marketed and installed, driving points are different:

- add to the system the largest amount of application oriented features
- generate many application oriented versions
- mantain and keep updated the system software.

It is quite clear that these aspects are influencing the way development steps are performed (technology driven implementation and market driven implementation can be handled with different approaches), but the integration/distribution phases are heavely influenced, with heavy feedbacks on the organisation of production process.
A possibility is to change the organisation structure indicated in fig. 3.1 into a structure in which the implementation groups are organised in a more product oriented view. A proposal is shown in figure 7.1.

The total elapsed time in order to obtain a product in the suggested process is longer then the one shown in fig. 3.1, but the structure has more intrinsic parallelism. As soon as the Basic System SW exists, developments for different market sectors can proceed and be distributed in parallel. Furthermore, while the basic system sw is working on the Nth release, developments for different Market Sectors, based on (N-1)th release, can be performed and distributed.
Finally, two feedback cycles are included; the first one-let call it technology driven - targeted to keep updated the basic System Software and the second one - let call it product driven - targeted to a fast reaction to market requirement. The second one can be significantly faster than the one in fig. 3.1, in which technology and product feedbacks are mixed and therefore the product feedbacks can be conditioned or even de-prioritized, in order to support apparently more urgent technological requirements.

In order to stress all the consequences from such production process organisation, it is interesting to structure a new software life cycle, that we can call the "Technology Market Compound Life Cicle" (See fig. 7.2).
Again, supported by the production process organisation shown in fig. 7.1, we have to pipelined phases with high degree of parallelism, which sharply separate the process of producing a new technologically advanced system and the process of releasing products driven by market requirement.
Important points to note on fig. 7.2 are:

- Architecture Definition is corresponding, at the technology level, to Product Definition at Market Level.
- The different steps of such life cycle may be described in greater details utilising one of the well known models (for example "Cascade" (BOE 81) or "Iterative"(BAL 84)), even if in this case we have sketched Baltzer's approach.

- Congifuration management is a very important phase also in technology driven development, where the fast prototiping technique is introducing a potentially unstructured development process.

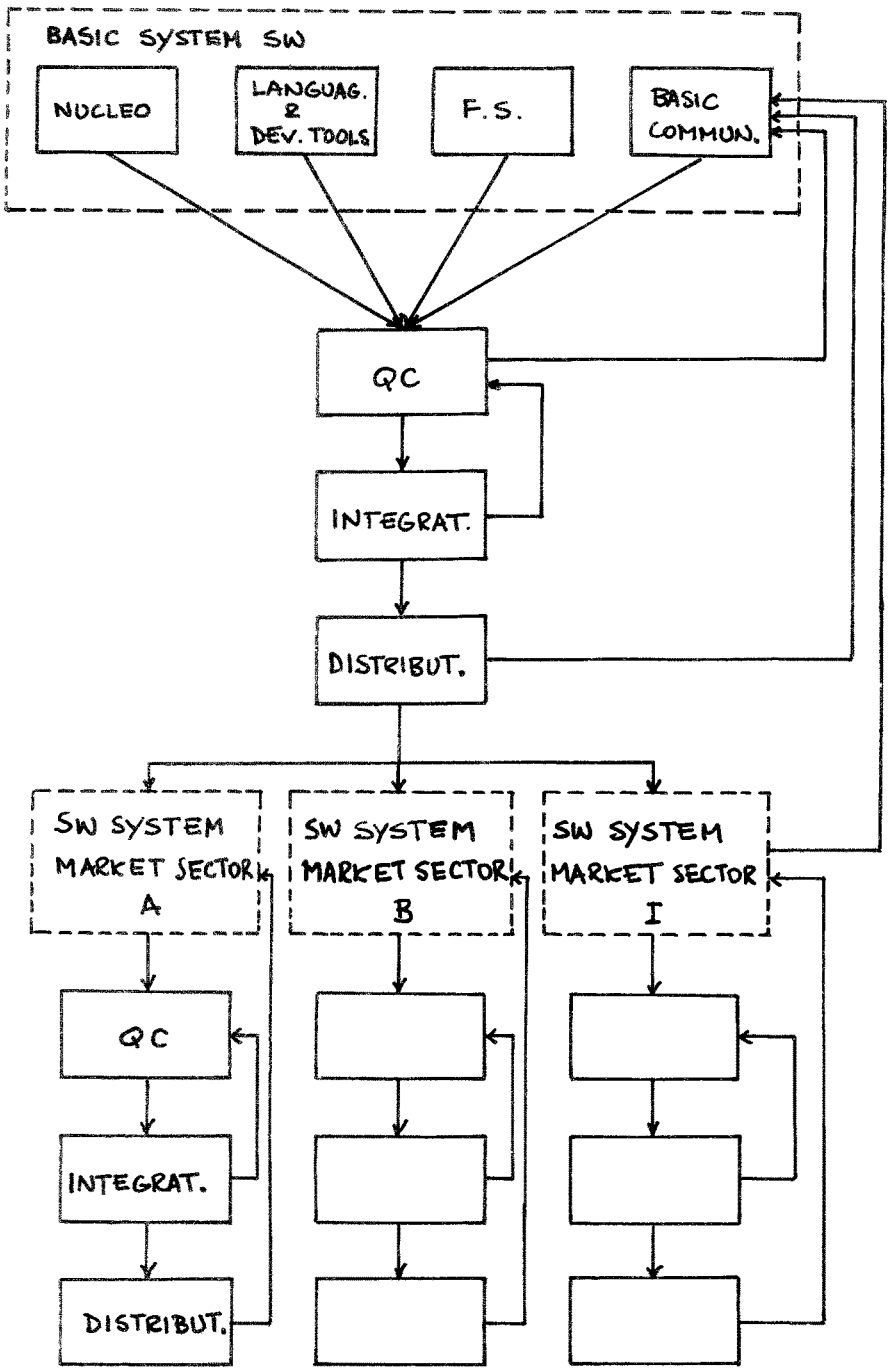

Fig. 7.1 Technology - Market Compound Production Process

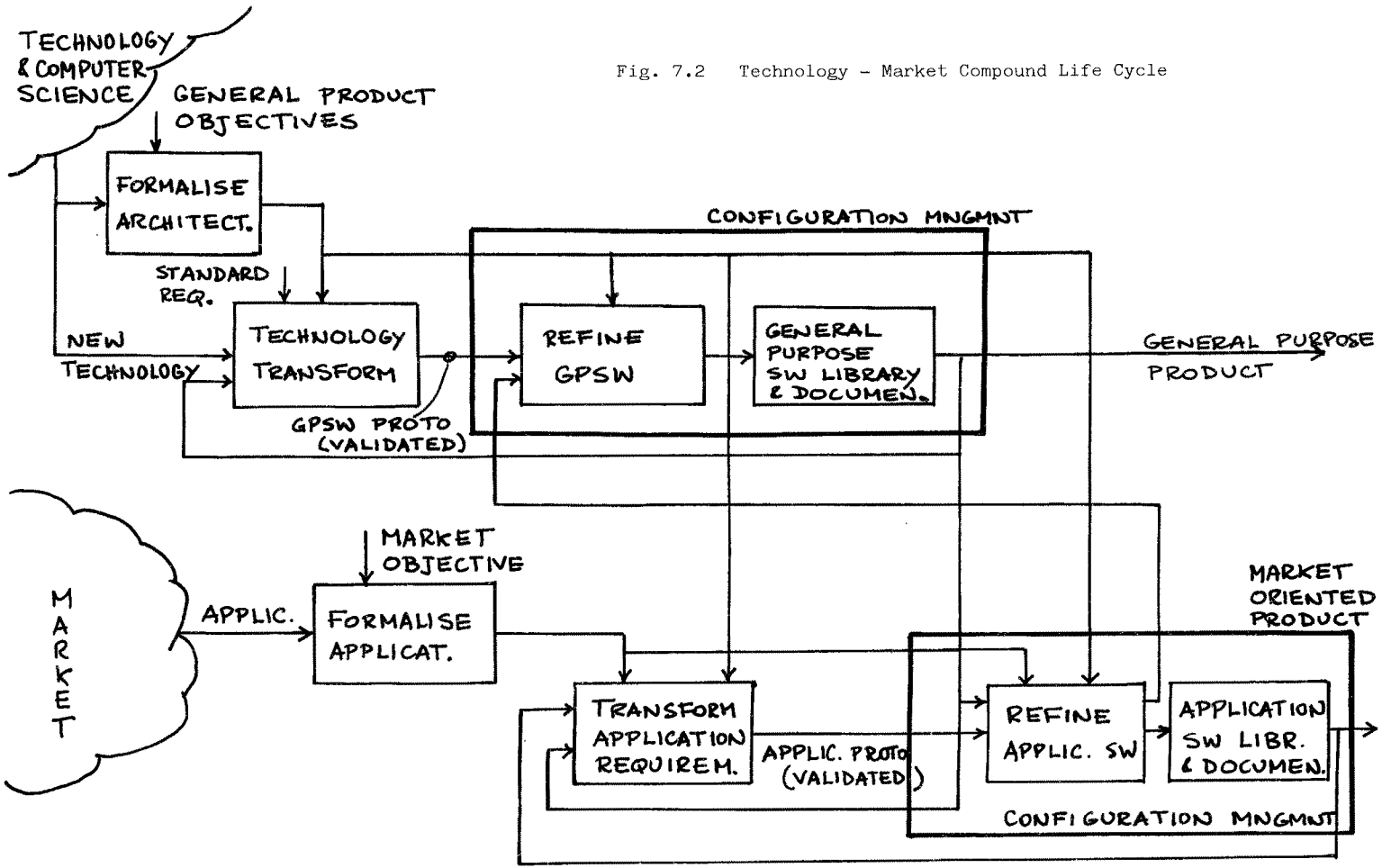

Fig. 7.2 Technology - Market Compound Life Cycle

Many parallel Market oriented cycles may be connected to one Technology cycles and therefore two different phases and distinct organisations are needed for Configuration Management.

8. CONCLUSIONS

Large enphasis is often given to programming tools, metodology and equipments, in order to increase software productivity. At Olivetti, starting a completely new project, a significant effort was dedecated to adopt modern and up-to-date tools and metodology. The project - "design and implementation of an entirely new general purpose multifunctional operant system and environment software for a minicomputer line" - found very beneficial the use of new tools up to the end of the first release.
A cascade software life cycle was adopted, with a large parallelism in the implementation fase and a single sequential integration, quality control and distribution phase.
When the software project is completely new and very complex, if the functional specs are well defined, this life cycle offers a strong control of the production process. It increases the probability of obtaing required functional characteristics while it offers lower control on delays and cost, because of lack of significant number of frequent feedbacks. Furthermore, the single integration quality control and distribution phase become to limiting, in order to exploit market potential of new system software while copying with it evolution.
A new process organisation, and software life cycle has been suggested in order to have a two phases process: one dealing with technologycal evolution and the other with market requirements. Each phase is supporting parallel development activities and offers many short feedback loops.
Because we are starting to experiment such approach, we are unable to report risults or criticisms.

BIBLIOGRAPHY

(BAL 84) R. Baltzer - Programming in the 1990's, Information Sciences Institute 1984.

(BOE 81) B.W. Boehm - Software Engineering Economics, Prentice Hall Inc, Englewood Cliffs.