Software Development: People, Process, Technology

Arash Khodabandeh, Paolo Palazzi
CERN - ECP Division - Programming Techniques Group

Abstract

The production of software is a labour intensive activity. Given the scale of software projects related to current and future HEP experiments, it is worth trying to improve the knowledge of the PEOPLE involved, the organization of the software development PROCESS and the TECHNOLOGY used in the various aspects of this activity. The goal is better systems at lower cost and happier users of the software.

This document can be accessed through WWW at:
URL = http://www.cern.ch/PTGroup/Papers/Sopron94/

The authors can be reached at:
[khoda@ptsun00.cern.ch] and [palazzi@ptsun00.cern.ch]

Published in the proceedings of the 1994 CERN School of Computing, Sopron, Hungary.
1 Introduction

Software is expensive. Building it and maintaining it are labour intensive activities, but delays in delivery can be very costly and any undetected problems may cause loss of performance and frustrate users. This also applies to High Energy Physics, even though physicists have been at the forefront of computing in the early days and, still are now in several areas.

Today the use of computers for physics is much more pervasive, and the size, complexity and lifetime of the software are much larger. For most of the physicists and engineers working in experiments, the business is physics or engineering, not computing, and many technical managers are not computer literate enough. The use of software is growing, and often it seems that the process of producing it is out of control. There is the feeling that not enough people around have the adequate software engineering background to improve the situation.

The software development teams around experiments are traditionally very independent. Aspects of the software development culture are local to various centers, although the same application packages are widely used in the community for some aspects of the work.

Three key components drive all improvements in software development productivity: the PEOPLE involved, the organization of the development PROCESS, and the TECHNOLOGY used. Talented people are a very important element in any software organization, but even the best professionals need an organized environment in which to do cooperative work. The same applies to advanced technology: without an organizational framework, it cannot be fully effective, and there is evidence that going for new technology instead of improving the process can make things worst.

2 People

It is obvious that any software system is dependent on the quality of the people who build it. People working in Particle Physics have been selected for their intellectual abilities, so nobody could argue that our software problems are due to the lack of talented people. Our people have produced many excellent programs and packages, and superior programs derive from superior design.

Superior design, on the other hand, is always the work of people who understand the application domain. This is a necessary condition of success, anybody who does not understand statistics cannot build a good statistical system. Of course he may profit from help by those who are familiar with technical aspects of software production. We can confidently assume that in our field we have bright people, who know their application domain. Let us see what they do in the scientific discovery process, and what is their involvement with software.

2.1 Software for HEP

Progress in the understanding of nature is the result of the combined work of theoretical and experimental physicists. Theoreticians put forward new theories compatible with existing experimental results, while experimentalists measure new physical phenomena to disprove the theories. To measure a physical process, large groups of scientists, engineers and technicians design, build and operate experiments and analyze statistically the large amount of data produced.

The production of software is involved at all stages of the process. Scientists can be more productive in a given domain or aspect of the data processing by using specialized systems or packages. Such software provides the appropriate abstractions through a user-friendly interface. Examples of that are Mathematica
and similar systems, CAVIAR, the histogramming package HBOOK and its more recent interactive version PAW, and so on.

### 2.2 Roles of people

Scientists who work close to the physics aspects of things will tend to use such specialized systems. These products will allow them to work effectively at the right level of abstraction in an easy-to-use context. Still there is a need to produce such programs, and this is the work of a minority of physicists. They are experts in a given technical domain, and at the same time act as specialized developers and build systems for the others to use. We refer to the authors of widely used scientific software, as well as to the physicists/programmers of a collaboration. There people take responsibility for producing the data acquisition, simulation, reconstruction and data analysis programs used by all the others.

In order to build such non-trivial software systems with a lifetime of several years or decades, specialized knowledge of software technology and a disciplined organization are both essential, and a few software engineering specialists need to be involved, who can advise on appropriate choices of organizational practices, methods and tools. Backing from the management is essential in enforcing such practices.

### 2.3 Open problems: specialization, training, recognition

Software for particle physics is a complex matter, where specialization is important, both in the application domain, and in aspects of software engineering. The various people involved act in complementary roles, these being:

- users of the software,
- producer of the software, or
- organization, methods and tools engineers.

It is important to recognize this situation, foresee the appropriate selective training for people doing different jobs, and the mechanism to acknowledge achievements and encourage mutual understanding. Teaching advanced software technology to those who do not need it may be confusing. Pretending that software is easy, so that whoever works in an experiment can stick his fingers in the code no matter how, is also inappropriate. This will create instability in the system, frustrate the software specialists, and generate animosity between “users” and “producers” of the software. (These considerations are controversial, and not generally accepted in the community.) Having said that, let us concentrate on aspects of the software production process, and ways to improve it.

### 3 Process

Many large organizations are actively trying to find ways of producing better software at lower cost, within predictable resource allocations and time estimates.

In recent years, it has been realized that the way to progress is to study and improve the way software is produced, while better technology only helps once the organizational framework is set. This discipline goes under the name of “Software Process Improvement” (SPI), and the reference for it is the work of the Software Engineering Institute [1] at Carnegie Mellon University.

The book by W. S. Humphrey, “Managing the Software Process” [2] is a good starting point to learn about the basic ideas behind this approach. SPI is based on statistical process control, defined in the ’30s and used
by Japanese industry after WW2, and by now in all industrial countries of the world. It is Total Quality Management applied to software.

### 3.1 The Software Life Cycle

#### 3.1.1 Building a house

Imagine that you just bought a nice piece of land, and would like to get a house built on it. You will live in the house, other people will design it and build it for you. So you begin thinking about what you need: your spouse likes gardening, you have an old car and want a garage where you can work on it, you are used to age wine, enjoy to sit by the fireplace, do not like the bauhaus style etc. You also note the constraints: shape and orientation of the land, composition of the family, size of your bookshelf, money available and so on. You write it all up on a couple of pages, and contact an architect recommended to you by reliable friends.

The architect listens to your wishes and constraints, and tries to define the properties that your ideal house should have. He will define number of floors and rooms, orientation of the driveway, size of the garage and so on.

When you are happy with what he proposes, he will actually design the house leaving out small details, and define its precise position and orientation. He will show you a sketch to make sure it is what you want, and produce drawings for the builders, as well as written specifications: type of bricks, colour of the walls and so on.

Now it is time to contact a building contractor, who will actually construct the house for an agreed cost in a given time. During the construction, details will be worked out between the architect and the builder. After a few months the house is ready, the architect will make sure that it corresponds to its design, ask for a few modifications, and soon the house is ready for you to move in.

You now live in the house with your family. You are happy, both the architect and the builder did a good job, without exceeding the predefined cost. There are a few minor problems with a leak in the roof and a few tiles in the balcony, but all this is readily fixed. This is not the end of the story: there will be regular maintenance of the plumbing system, chimney, drains, painting etc., but if design was right and materials good and well put together, all this will not be terribly expensive nor problematic.

What really will require substantial work at a later stage has to do with the fact that your needs change. A new baby comes, and you will have to split a large room in two, you buy a small boat and need to enlarge the garage to keep it there in the winter, and so on. The more of all this you were able to predict when you spoke to the architect, the easier and cheaper it will be to realize. Having the original plans available is also very helpful.

#### 3.1.2 Life cycle

The purpose of this story was to explain by analogy how non-trivial software systems are built today. The owner of the house had to go through several phases to get from a wish to a house he can live in. These correspond broadly to the phases of the software life cycle (figure 1), namely:

- User Requirements definition;
- Software Requirements definition;
- Architectural Design;
- Detailed Design and construction;
- Delivery to the user;
- Operations;

To consider that the life cycle phases are disjoint in time (waterfall model) is a simplification. Frequently the tasks of different phases are performed somewhat in parallel (concurrent engineering). It is however
important to distinguish them logically, and identify documents that are the outcome of the various
phases. For software that comes in successive version a, the life cycles of the versions usually overlap: one
may be designing version 3 while delivering version 2 to users who run version 1.

3.2 The Software Process

A process is the set of actions, tasks and procedures involved in producing a software system, throughout
its life cycle. A ”good” software process must be predictable: cost estimates and schedules must be met,
and the resulting product should be robust and offer the required functionality.

The objectives of software management are to produce according to plan and to improve the capability to
produce better systems. The basic principles are those of statistical process control [3], used successfully in
many fields of industry. A process is under statistical control if its future performance is predictable within
established statistical limits. The cornerstone of statistical process control is measurement, and measuring
the software process is not easy. One needs to define the appropriate reliable metrics, provide the
infrastructure, and consider that people are involved and may resent being ”measured”.

3.2.1 Software Process Model

A typical process model covers the whole of the life cycle and clearly defines each phase and the tasks and
activities to be performed within that phase as well as those activities that are continuous throughout the
life cycle (see figure 1).
We discussed phases already in 3.1.2, the most important activities are:

- Project management: planning, resource allocation, tracking, deadlines;
- Verification and Validation: reduce software defects and make sure that it does what the user wants;
- Software Configuration Management: handle the actual code, variants, versions and so on;
- Maintenance: fixing bugs, coping with changes at the boundary of the system, adding new functionality.

An example of a software process standard is ESA PSS-05 [4], defined by the European Space Agency (ESA) to fulfil its own needs to produce safety critical software. It has recently been used in HEP also, and we will cover aspects of it in the next chapter.

### 3.3 Software Process Maturity

The quality of a process can be measured in terms of maturity against a recognized framework. The higher the level of maturity the higher the quality of the organization’s process. The reference is the Capability Maturity Model (CMM) [5] [6] proposed by the Software Engineering Institute (SEI) [1] (see figure 2).
The SEI maturity framework consists of five levels:

1: **the initial level** is where most organizations begin. A level one process is usually ad hoc, few processes are defined and success depends more on individual effort than team work;

2: **the repeatable level**: basic project management processes are implemented to follow cost, schedule and functionality. The process is sufficiently defined to be able to repeat earlier successes on projects with similar applications;

3: **the defined level**: the processes for both management and engineering are documented, standardized and integrated into a standard organizational software process.

4: **the managed level** requires that detailed measures are taken of the process and product, and that process and product are quantitatively understood and controlled.

5: **the optimizing level**: continuous process improvement is in place, using quantitative feedback.

The most difficult step is to move from level one to level two because of all the management procedures and activities that have to be put in place. At higher levels of maturity the project management has a better visibility on the software process, and can take corrective actions. This visibility enables to take corrective action sooner when something goes wrong. The difference in maturity levels is also characterized by the ability to accurately estimate and schedule product delivery and the capability of the process to meet those estimates. As an organization moves up the maturity levels, estimates become more accurate and the time required to produce a system is shorter.

### 3.3.1 Assessment

Before a successful attempt can be made to improve the software process, it is essential to carry out an assessment in order to rate the current status, formulate goals and an improvement program to reach those goals. The Bootstrap Project [7] by ESPRIT in Europe has been in operation for a few years. Starting from an
extended SEI model, an assessment procedure has been defined, and tried out on a number of organizations in different countries. The assessment is based on two questionnaires, one addresses software quality attributes while the other deals with management issues that reflect the organization’s maturity. The replies are then analyzed and the results are presented showing the key attributes grouped under three main areas: Organization, Methodology and Technology.

The weakest aspects will point in the direction of improvement actions that must be identified by the assessment team and given a priority that takes into account also external conditions such as cost, resource availability etc. Actions must be followed up, and the assessment/actions cycle repeated regularly, if the organization intends to improve its maturity. A statistical survey of results from different types of organizations and countries has shown that most of the European software producers are between levels 1 and 2. Telecom companies are almost at 2, banks just above 1. National differences are also visible.

3.3.2 Is it worth improving? How?

By now the practices of SPI are well established, and have been applied in a large number of organizations for several years [8] [9]. The results prove unmistakably that the economical benefits are largely worth the investment. Spending about 1.5K ECUs per software engineer per year, can yield gains in productivity between 10 and 70%. Early defect detection, time to market, and quality also improve, to the point that the return on investment for SPI is about 500%.

Experience suggests that necessary conditions for success of an SPI effort are:

- full backing of the top management;
- implication of everybody involved;
- provision of adequate resources;

We believe SPI is worth trying also for HEP software, with an appropriate strategy that takes into account the peculiarities of our development environment.

4 Technology

Various activities and tasks are performed during the software development process to go from user requirements to the delivered product. The machine can often help the human, sometimes even automatically, to avoid errors, work faster and concentrate the attention at the conceptual level.

The programs that help building software are called software development tools and are very specialized. Some are designed for one activity during the whole life cycle, while others for a specific action limited to one phase of the life cycle. In these notes we cannot cover all aspects of the technology concerned with software development. We illustrate our presentation with a few tools that can help to introduce the concepts and that we have experience with.

4.1 Technology for Life Cycle Activities

4.1.1 Project Management

Some tools are intended to support individual programmers working on specific tasks (4.2), while others help the management and organization of a project. Some are commercial products and one needs to pay for them, others are public domain software. ProcessWeaver (figure 3) is a process management tool that helps to automate the development process in terms of tasks, dependency between tasks and input.
and output of each task. Once the process is modelled, ProcessWeaver helps to follow its execution by automating the start of each task within the process when all required input available. It can also start the appropriate tools needed for the execution of the task. On the other hand, MS-Project (figure 4) is a project management tool to handle resources allocated to the project in terms of manpower, time and other resources.

figure 3 ProcessWeaver organizes the workflow of a group of developers. Each project member has an agenda (2) where each icon represents a task he/she must perform. Clicking on an icon opens a work context (3) with a description of the task (3a), icons for all input and output documents to be produced (3b 3c) and completion/decision buttons (3d). Clicking on an input/output document will start the appropriate tool (e.g. FrameMaker, C/C++ programming environment ...). Similarly, clicking on a completion/decision button will end the task and send it to the right project member. The manager uses PetriNets (1) to model the process describing the tasks to be performed, by whom, in which order and the input and output to each task. Each place (1a) models a task under execution by one or more project members. Each transition (1b) models the submission of a task to one or more project members after the completion of the previous task(s). The member of the team can open a read-only dynamic view of this process model to consult its state and see which are the tasks in progress and who is handling them.
The MS-Project window is divided into three sections. Section (1) is a list of tasks and sub-tasks with the start time, end time and the resources allocated to them. Section (2) shows a time flow of all tasks with their dependencies and milestones. Section (3) shows for each resource when it is in use, free or over allocated. Contrary to ProcessWeaver (figure 3), MS-Project only displays the information statically: it must be fed with information on the state of each task, and it reschedules all the dependent tasks automatically.

ProcessWeaver and MS-Project are complementary and can be coupled: ProcessWeaver can read the MS-Project description file to generate the process skeleton, and MS-Project can read runtime information on the process status generated by ProcessWeaver.
Another domain where tools help is the **software process assessment**, to measure the maturity of the process and set improvement goals. The ESPRIT project, AMI (Application of Metrics in Industry), produced a 12-step method for the measurement of the software development process and starts with an SEI self assessment. It ensures that quantitative approaches are used to achieve company objectives and improvement in the software development process. AMItool (figure 5) implements the AMI method and takes the user from the SEI questionnaire to the measurement plan.

![AMItool](image)

**figure 5** AMItool implements the AMI method. To begin with one answers an SEI questionnaire (1). AMItool compiles and presents the results of the questionnaire in a Kiviat graph (2). On this Kiviat graph each axis represents one domain of the process (e.g. Training, Organization) and is divided into the 5 SEI CMM levels. A dot is placed at the corresponding level of maturity on each axis. The process manager, using the results of the questionnaire, identifies primary goals (e.g. ‘Gain Better Understanding of Project Costing’) and decomposes them, in the goal tree (3), down to basic sub-goals. He then defines metrics and interpretation rules and associates them to goals and basic sub-goals (4). AMItool then generates a plan (5) for the measurement of the process. After a while, another AMI iteration can start.
Communication and information flow in a project can be improved by suitable tools. For example, a discussion involving more than two people over the network, can be difficult to follow because more than one idea is discussed at the same time. This is where WIT (World Wide Web Interactive Talk) \[11\] can help. WIT (figure 6) allows discussions through the WWW and displays discussion items in a structured fashion. Lotus Notes is a commercially available tool addressing the same problem, that works across UNIX machines PCs and Macs.

A WIT system is composed of Discussion Areas. A Discussion Area is a general subject, which has specific aspects, called Topics (1), and each Topic is composed of Proposals (2) which generate an exchange of messages (3). The big advantage of WIT over the news groups is the structuring. When looking at a Proposal one can see what has been discussed, who answered to whom, who agrees/disagrees with whom, etc. WIT is not only useful to browse through discussions but it is also possible to agree (✓) or disagree (✗) with any statement in the Proposal using the tool itself.
At each stage of the life cycle many documents are produced by different people: requirements specification, analysis, source code, user manuals, etc. These documents are heavily structured and cross-referenced. LIGHT (LIfe-cycle Global HyperText) [12] integrates these documents and allows to navigate (figure 7) between and inside requirements, diagrams, code, manuals, etc.

**figure 7** The LIGHT concept is illustrated by this ADAMO [13] example. The navigation map (1) shows all documents that are linked here: pictures, requirements, analysis diagrams (2), DDL (3), FORTRAN source code (4) and ADAMO reference manuals (5). Clicking on any of the document symbols will go to the corresponding document, which is itself linked to the others. For example clicking on the “track” class in the diagram will open the “track” declaration in the DDL. Similarly clicking on an ADAMO routine call within the code will open the ADAMO manual entry for that routine.

### 4.1.2 Verification and Validation

PSS-05 defines the two sub-activities in the Software Verification and Validation (SVV) activity [4] [14] as follows:

“Verification is the act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether or not items, processes, services or documents conform to specified requirements (ANSI/ASQC A3-1978).

Validation is, according to its ANSI/IEEE definition, ‘the evaluation of software at the end of the software development process to ensure compliance with the user requirements’. Validation is, therefore, ‘end-to-end’ verification.”
The Verification software must itself be specified and written. Many commercial tools are available to help organize verification and validation activities. For example, Purify and TestCenter (figure 8) detect memory access errors and memory leaks in C and C++ programs. Similarly, TestCenter and Logiscope (figure 9) can visualize which part of the code has been executed during tests (test coverage). Programming in a higher level language is another way of preventing this class of programming errors.

**figure 8** TestCenter is an environment for C/C++ testing and debugging. It shows memory leaks (memory with no pointer to it) and possible memory leaks (memory with a pointer to its middle). It can also detect runtime errors such as: pointer and array boundary errors, use of uninitialized memory, attempted write to read-only memory, errors using free and unanticipated termination signals. For both the memory leaks and runtime errors, TestCenter gives respectively in the leak browser (1) and the error browser (2) information on the problem. It also shows the source code line where the error was introduced, and can help with test coverage analysis to some extent.

**figure 9** Logiscope performs test coverage analysis for programs. It displays the call graph tree (1) and highlights the function calls not executed; similarly for each function, it shows the control graph (2) and highlights the instruction blocks not executed.
The verification and validation will introduce changes and new releases of the software. It is important to be able to perform defect logging and tracking with a flexible source code manager with a database and a query system.

4.1.3 Configuration Management

The elements subject to configuration management are: source code, internal documentation of all sorts (requirements, etc.), manuals, tools, test software and test results. The software configuration management includes many tasks such as: 1- controlling the release and changes of each element 2- logging and analyzing changes and reasons for each of them.

As mentioned in 4.1.2 a source code manager based on some kind of a database can help control who changes what, why and when (librarian checkin/checkout). It must also log all the changes, provide statistics and/or process queries.

The last configuration management task consists of verifying the completeness and correctness of single modules. For example, one must check the style and do statistical measurements to make sure that the code follows agreed criteria. This work can be done with specialized tools, possibly integrated in the source code manager. Logiscope that we saw as a test coverage tool in figure 9 can compute software metrics for various programming languages and display the results graphically (figure 10).

4.1.4 Maintenance

Once the software is developed, tested and delivered, users will start to work with it, and suggestions and bug reports will come in. The maintenance team will sort this input and set priorities. After improving the
software the development team carries out regression tests (i.e. old tests still work and no new bugs are introduced). They may also have to introduce new test programs for the newly detected bug.

4.2 Technology for Life Cycle Phases

In the early phases of the life cycle, user requirements must be expressed in detail. This is usually done with structured text in a natural language, and some kind of referencing scheme. A good text processing system with templates and cross referencing can be very useful, but specialized requirements tools are beginning to appear.

4.2.1 User Requirements

This is how ESA PSS-05 [4] [15] defines the User Requirement (UR) phase:

“The UR phase is the ‘problem definition phase’ of a software project. The scope of the system must be defined. The user requirements must be captured. This may be done by interview or survey, or by building prototypes. Specific user requirements must be identified and documented in the User Requirements Document (URD).

The involvement of the developers in this phase varies according to the familiarity of the users with software. Some users can produce a high quality URD, while others may need help from the developers.

The URD must always be produced. The review of the URD is done by the users, the software and hardware engineers and the managers concerned.”

As an example of URD, one can have a look at WebMaker [16] and WebLinker [17] where PSS-05 was used for expressing the User Requirements.

4.2.2 System Requirements

ESA PSS-05 [4] [18] defines the System Requirement (SR) as:

“The SR phase is the ‘analysis’ phase of a software project. A vital part of the analysis activity is the construction of a ‘model’ describing ‘what’ the software has to do, and not ‘how’ to do it. Building prototypes to clarify the software requirements may be necessary.

The principal deliverable of this phase is the Software Requirements Document (SRD). An SRD must always be produced for every software project. Implementation terminology should be omitted from the SRD.”

The WebLinker System Requirements [19] have been written using the PSS-05 standards. They contain the traceability matrix that proves the SRD’s completeness: System Requirements cover all User Requirements and nothing more.

To formulate the User and System Requirements according to a given framework, one may use specialized tools such as DOORS. These tools check the consistency between Requirements and automatically maintain traceability information. The Requirements mentioned above and in 4.2.1 have been written using simple templates for a text processor and automatically translated to WWW [20].

4.2.3 High-Level Design

During this phase (called Architectural Design in PSS-05 [4] [21]) the overall structure of the system is specified: the software construction begins. There are several methods, more or less object-oriented, that can be used in this activity such as SASD/TR [22] [23], PROTOB (PetriNets with objects) [24] [25], Class Relation[26], Martin/Odell [27], OMT [28], FUSION [29] to name just a few. They all have different models to represent different aspects of the system. For example OMT and Class Relation put the emphasis on the object model, while PROTOB has a stronger dynamic model and SASD focuses first on the functional model. The choice of a method is driven by the characteristics of the system under development, knowledge of the team and so on.
A CASE tool can help in the application of the method. For example, Artifex (figure 11) supports the PROTOB method based on High Level Colored Concurrent PetriNets with simulation and C/ADA code generation. Similarly, Objecteering (figure 12) implements the Class Relation method with C++ code generation. For a given method one finds usually many tools to support it and needs to choose the appropriate one. IEEE has produced recommendations for the evaluation and selection of CASE Tools [30] and suggests criteria such as: how well the methodology is supported, are all the concepts covered, how much consistency checking is performed, how much the tool can be customized and the quality of its graphical user interface.

figure 11  Artifex checks that the PetriNets are consistent and complete. It also allows debugging and prototyping with its interactive simulation tool. Artifex generates distributed code along with the Makefiles. Developers can even compile, start the remote code and monitor the execution using the interface. The logic of the PetriNet and the inter-process communications are all handled by the tool. Developers can concentrate on the model and specification of the actions in a transition.
With Objecteering, the developer performs the analysis using the CR notation (1). He can enter the method specification in C++ through the graphical user interface (2). Objecteering can then generate the C++ code and automatically integrate the user specified part in the right place (3). All the syntax of the C++ files, the test statements (IFDEFs) and the relationship operators are automatically generated and taken care of by the tool. Developers can concentrate on the method and the model specification.

4.2.4 Detailed Design and Coding

This phase (called Detailed Design and Production in PSS-05 [4] [31]) uses as input the High-Level Design phase output and produces the software. This code is then unit tested, integration tested and system tested. The choice of the CASE tool to support the activities of this phase is driven mostly by the language used and the detailed required functionality. A programming environment with a class/inheritance browser is useful for Object-Oriented languages. For example, ObjectCenter (figure 13) is a complete C/C++ programming environment with a debugger, an interpreter and class/function/data browsers. One may also employ a testing tool that can detect memory leaks if dynamic memory is used (see 4.1.2 figure 8).
ObjectCenter is a complete C/C++ programming environment. Its main window (1) has a source code browser area (1a) and an input area (1b). The input area can be used as a C/C++ interpreter where all the data types and classes defined in the source code are accessible. The error browser (2) can detect loading/compilation and run time errors. Clicking on the error will highlight the error-prone line in the source code area. The inheritance browser (3) can display a class tree. The programmer can browse through all the classes/structures defined in the loaded code and defined in the external libraries. Selecting a class from the inheritance tree will display the class browser (4) where all the data and functions defined in the class or inherited are listed. The programmer can filter on attributes (e.g. private, public, static, virtual ...). Selecting a function from the class browser will display the cross-reference browser (5) where the programmer can navigate in the call tree and identify the callers/called functions of any function. The programmer can use the main window to run the program, set breakpoints and step through it. At run time the programmer can display an object in the data browser (6) and see the values of its fields. If the object points to another object then clicking on the pointer record next to the address field will display the pointed object, building dynamically the data network. During execution the displayed object are dynamically updated. The browsers presented here are callable in any order and from one another through handy pop-up menus.

There exist other development environments such as VisualWorks for SmallTalk and LispWorks for LISP with objects. Some of these can be combined with testing tools. For example, TestCenter and ObjectCenter can be coupled to form a complete environment for editing, browsing, testing and debugging C/C++ programs.

4.2.5 User Manuals

An important deliverable, beside the software itself, is the documentation for the user. This is true for an interactive software with a GUI (graphical user interface), but also for a library used by other programmers. A User/Programmer’s Guide, a Reference Manual and often a Tutorial are part of the documentation. These documents are highly structured and can be written following precise guidelines.
such as STOP [32]. They usually include a table of contents, a map of the document, a glossary, an index and a bibliography. Graphics is a must to make the document easier to understand and more readable.

Often the same information can have many incarnations that must be kept coherent. For example a Reference Guide that exists on paper can also be published electronically on the World-Wide Web [20] and be integrated in the on-line help. The paper “An Automated System for the Maintenance of Multiform Documents” [33] (figure 14) shows how this was achieved for one specific product.

![Figure 14](image)

As an example, the paper “An Automated System for the Maintenance of Multiform Documents,” shows how the ADAMO documentation is automatically maintained on paper, World-Wide Web and KUIP. The system uses FrameMaker (1) as the master format and to produce the printed version. Automatic converters generates HTML for WWW (2) and CDF for KUIP (3). Examples are dynamically compiled, run and included, to ensure that they are inline with new features.

### 4.2.6 Integration

The integration phase is where all the pieces of the system are put together. These may consist of different executables, libraries, include files, documentation (see 4.2.5) and the installation scripts. The integration is a detailed process where deliverables must be put in the right place and tested. The failure of any test leads to further development followed by another integration phase. This means that one may need to go through the integration process many times and follow the same steps each time. This is a typical example where a tool such as ProcessWeaver can help in automating the repeatable part of the process. As mentioned in 4.1.1 figure 3, ProcessWeaver is a management tool that helps to automatize the process (figure 15) in terms of tasks, dependency between tasks and input and outputs of each task.
Like Artifex (see 4.2.3) ProcessWeaver uses PetriNets for modelling. The above model is the “WebMaker TAR file verification process model.” The “TE_TAR_Collect_TS_TAR_Creation” is activated when a previous process (not shown) leads to the collection of all deliverables. Then the TAR file is created. With the transition “TE_TAR_Creation_TS_TAR_Review” all testers concerned are asked to test and accept/refuse it. Depending on the number of refusals “CollectAnswer” will either go back to produce a new tar file with “TAR_Review_Incomplete” and restart the process or continue with “TAR_Review_Complete_TS_TAR_MAIL” to mail the announcement of the new version to the user. The actions performed in each transition (creation of the TAR, test of the TAR, delivery of the mail) are not modelled as part of the process, but ProcessWeaver can call external tools. For example it can call a shell script for the creation of the TAR or to run tests and analyze the output.

4.2.7 Delivery

Once the TAR file of deliverables is ready the delivery can start. It may consist of several steps (figure 16): inform the user of the availability of the software, distribute it, provide installation instructions and finally run the installation scripts. Nothing magic, but a non negligible amount of work.
The files above are from WebMaker, a current development of the CERN ECP Programming Techniques Group. The availability of the software is announced to the users registered on a mailing list via a Bulletin (1). The user may ask to retrieve the software by ftp and is given instructions by E-mail (2). In the delivery kit, README files (3) explain the contents of each file/directory and lead the user through the installation process. A large part of the installation is done with shell scripts (4).

4.2.8 User Support

Part of user support is information. The goal is to allow users to work without assistance by providing them with all the necessary information that is constantly kept up to date. The World-Wide Web is ideal for this task (figure 17): the information is easily accessible, it can be linked to other documents and users get
always the latest versions. Some of the WWW information can be derived from the printed version (4.2.5 figure 14).

![WebMaker information Web](image)

**figure 17** This is the WebMaker information Web. From the entry page (1) WebMaker users can consult the User’s Manual (2) (generated as explained in 4.2.5), the Frequently Asked Questions (3), interesting examples, list of bugs and suggestions to name but a few.

Documentation and public information will never be perfect and usually many user’s queries will need to be handled preferably via electronic mail. A system such as MH-mail (figure 18), a highly configurable mail handler, can be of great help by filing messages automatically.
Acknowledgments

We would like to thank Ian Hannell for contribution and advice concerning the process part and help with this document, Pierrick Pinasseau for setting up the cluster we used in Sopron and for his daily assistance during the school, Joel Closier for the installations of all the software tools used, and the suppliers of the tools who kindly provided us with free demonstration licences and information leaflets for those who attended the lectures.

References


figure 18 MH-mail and its X interface EXMH can file the mail in a separate folder (2), reply to the mail, register the sender in a mailing list etc. All this according to rules specified in a configuration file and depending on the subject, sender or any header information in the mail. It also acts as a standard mail tool with a GUI (1) to read messages, and reply manually to them.
Schlumberger Laboratory for Computer Science, Austin, Texas, USA


[12] Lifecycle Global HyperText (LIGHT), http://www.cern.ch/Light/

Techniques group, http://www.cern.ch/Adamo/ADAMO_ENTRY.html

Agency, 8-10, rue Mario-Nikis, F-75738 Paris Cedex.

Agency, 8-10, rue Mario-Nikis, F-75738 Paris Cedex


Space Agency, 8-10, rue Mario-Nikis, F-75738 Paris Cedex.

[19] WebLinker System Requirements,
http://www.cern.ch/WebLinker/WebLinker_21.html#HEADING20


Space Agency, 8-10, rue Mario-Nikis, F-75738 Paris Cedex.


Englewood Cliffs, N.J.


[25] ARTIS, AXEN-C-SCR-MAN-E-V2.3. Artifex C Script REFERENCE MANUAL, ARTIS, Torino, Italy,


[29] Object-Oriented Development, The Fusion Method, Derek Coleman et al., Prentice-Hall, 1994,


Space Agency, 8-10, rue Mario-Nikis, F-75738 Paris Cedex.

[32] Everything You Ever Wanted to Know about Documentation (and Should Have Asked), C. Fidge, D.
Heagerty, Telecom Australia, 1986

[33] An Automated System for the Maintenance of Multiform Documents, B. Rousseau, M. Ruggier, M.
Smith, CERN/ECP Report 94-14,
http://www.cern.ch/WebMaker/examples/CHEP94_adamodoc_2/www/adamodoc_1.html