

Effort profile & scheduling implications of ESA's SW eng. standards

Written by	L.Tuohey, School of Computer Applications Dublin City University
Reference & Version	<u>DCU.LT-G1.001-WPAPER-002v1.1</u>
Date of issue	<u>November 14th 2000</u>

CONTENTS

Abstract.....	6
1. Introduction.....	7
2. ESA Software Engineering Standards	8
2.1 Outline of the standards	8
2.2 The standards as a basis for project definition.....	9
3. Some Central Schedule Considerations	10
3.1 Implications of the ESA standards	10
3.2 Importance of choice of life cycle model	12
3.3 Impact of errors detected during testing.....	14
4. Key Parameters of Software Projects.....	15
5. Investigation of Model Projects.....	17
5.1 Approach.....	17
5.2 Selection of model projects	17
6. Presentation of Results	19
6.1 Model 1	19
6.2 Model 2	21
6.3 Model 3	23
6.4 Model 4	27
6.5 Model 5	27
6.6 Model 6	31
7. Discussion of Results	33
7.1 An overview	33
7.2 Effect of product size.....	33
7.3 Effect of project category	35
7.4 Impact of errors detected during test.....	36
7.5 Choice of life-cycle model	38
8. Conclusions.....	38
Acknowledgement.....	39
9. References.....	39
Appendix: Some Modelling Details.....	41
A.1 Production of a software unit.....	41
A.2 Impact of error detected at unit test.....	42

A.3 Impact of error detected at integration test.....	42
A.4 Impact of error detected at system test	43
A.5 Internal review process	44
A.6 Defining SW requirements and architecture	45

Abstract

ESA's Software Engineering Standards (PSS-05-0), and other similar standards, specify the elements of software projects, their phases, the technical activities within each phase, the review processes, the documentation, the associated managerial work, configuration management tasks, and so on. This paper offers an approach to synthesising these several elements with the objective of assessing impact on project planning, particularly in terms of effort expenditure profile and schedule. A number of representative model projects are analysed and implications for practice are noted.

1. Introduction

According to Yourdon¹⁴ 'The mathematics of the relationships between X[number of people], Y[units of time], Z[cost], P[units of functionality], and Q[bugs per function point] are something we don't know enough about at our present level of software engineering'. A basic goal of the present paper is to contribute to a better understanding of some of these relationships. The approach taken is to perform a detailed examination of the anatomy of software projects as embodied in a well-established industry standard, namely ESA's software engineering standard.

ESA's software engineering standards are 'applied for all deliverable software implemented for the European Space Agency (ESA), either in house or by industry'(p. ix). According to ESA's Technology Transfer Group², the latest version (Issue 2) of the standards has attracted great interest outside of the Space sector. The attraction for external organisations is that they can 'buy in' to a product which has been shown to work in practice.

The ESA software engineering standards may be classified as procedures to follow within a specific project (p. 48³). There are similar standards in, for example, civil aviation²⁰, defence^{21, 22}, general safety-related software²³, medicine²⁴, and the nuclear industry. The IEEE software standards are also rather similar and indeed are explicitly stated to have been taken into account for the issue 2 of ESA's standards(p. viii). Thus, while this paper considers specifically the case of ESA's standards the same approach could be applied to other, similar standards.

The foregoing class of standards are to be distinguished from standards on quality systems, notably ISO 9001, which are not project specific(pp 47-49³). However, the ESA software engineering standards may certainly be applied on projects within a system which meets ISO 9001. Hence, the methods and results reported here are intended to be a contribution to modelling of software development processes and environments in general.

ESA's Software Engineering Standards specify the software development process in some detail. They define rules and guidelines to be observed to assure a satisfactory product in terms of technical scope and quality. There are associated implications on cost and schedule which should be appreciated by both software acquirer and developer. This is particularly important when the software is embedded in a wider, concurrently-developed system.

It is important that the level of detail of the standards should not be a barrier to identifying key issues and decisions, nor to non-specialists in software engineering. Thus, it is desirable to have an approach which ensures that a project properly reflects the complexity of the standards while at the same time providing appropriate high level perspectives and summaries.

The content of the paper is as follows.

Key attributes of ESA's Software Engineering Standards are noted in section 2, with particular emphasis on their implications for project definition. Some central considerations for project scheduling are introduced in section 3, particularly the consequences of reviews, the choice of life cycle model (waterfall or more complex), and the impact of errors detected during test.

Each software development project is unique with its own particular technical and programmatic requirements. However, for the purpose of analysis, it is very convenient to identify parameters or characteristics common to all projects. This allows projects to be classified according to characteristic values. The characteristics - parameters and factors - of interest for this paper are identified in section 4.

Based on the characterisation of section 4, a number of representative model projects are selected (section 5) for subsequent analysis. The method of analysis consists of application of a general parametric tool (prototyped by the author at CAPTEC Ltd in 1993-1994) for process modelling. This tool has been interfaced to Microsoft Project whose facilities are used to derive and display results for each model, with particular emphasis on effort expenditure profiles and schedules.

The results obtained are presented in section 6, followed by a discussion (section 7) in which the main findings are highlighted. Some practical implications and guidelines, suggested by the results, are noted. Finally, in Section 8, conclusions are drawn and possible directions for further work are outlined.

The appendix provides details on how some basic activities are incorporated in the project models.

2. ESA Software Engineering Standards

2.1 Outline of the standards

The purpose of this paper is not to provide a complete description of the ESA software engineering standards but rather to focus on practical implications of applying them. However, as a preliminary, an overview of the standards is presented in Figures 2.1-1 to 2.1-3.

Part 1 PRODUCT STANDARDS: Standards, recommendations, guidelines about the product (software) to be defined, implemented, operated and maintained.

Part 2 PROCEDURE STANDARDS: Procedures used to manage a software project.

Part 3 APPENDICES: Glossary, Software project documents, Document templates, Summary of mandatory practices, Form templates.

Figure 2.1-1: High-level structure of ESA SW Engineering Standards

1. SOFTWARE LIFE CYCLE 2. USER REQUIREMENTS DEFINITION PHASE 3. SW REQUIREMENTS DEFINITION PHASE 4. ARCHITECTURAL DESIGN PHASE 5. DETAILED DESIGN & PROD'N PHASE 6. TRANSFER PHASE 7. OPERATIONS & MAINTENANCE PHASE where each phase (2 to 7) is further subdivided into X.1 Introduction X.2 Inputs to the phase X.3 Activities X.4 Outputs from the phase	1. MANAGEMENT OF THE SW LIFE CYCLE 2. SW PROJECT MANAGEMENT 3. SW CONFIGURATION MANAGEMENT 4. SW VERIFICATION & VALIDATION 5. SW QUALITY ASSURANCE where each process (2 to 5) is subdivided into X.1 Introduction X.2 Activities X.3 The <process> Plan X.4 Evolution of the <process> plan throughout the life cycle
---	---

Figure 2.1-2: PRODUCT STANDARDS part of ESA SW Engineering Standard

Figure 2.1-3: PROCEDURE STANDARDS part of ESA SW Engineering Standard

2.2 The standards as a basis for project definition

Basic steps in defining any project are (compare Hamilton¹⁶, for example)

- A. ESTABLISHING A SET OF TASKS
- B. SPECIFYING SEQUENTIAL RELATIONSHIPS BETWEEN TASKS
- C. DEFINING AN HIERARCHICAL WORK BREAKDOWN STRUCTURE - WBS
- D. ALLOCATING RESOURCES TO TASKS
- E. ASSIGNING DURATIONS TO TASKS

Steps A and B are automatically defined, for the most part, for projects on which the ESA software engineering standards (or similar) are applicable. Also, many of the elements of step C are implied by the standards (though there is some flexibility). For example, the standards require the activity breakdowns depicted in Figure 2.2-1.

<i>DEFINE USER REQUIREMENTS (pp 1-13, 1-14)</i> 1. Capture user requirements 2. Determine operational environment 3. Specify user requirements [follows 1, 2]
<i>DEFINE SOFTWARE REQUIREMENTS (pp 1-20, 1-21)</i> 1. Construct logical model 2. Specify software requirements [follows 1]
<i>DEFINE SOFTWARE ARCHITECTURE (pp 1-30, 1-33)</i> 1. Construct physical model 2. Specify architectural design [follows 1]

Figure 2.2-1: Activity breakdown required by ESA standard

The standards impose some general constraints on resources (Step D), especially on the role and responsibilities of different participants. Implications on step E are addressed in section 3.

Naturally, it will still be necessary to tailor planning to take account of project-specific factors. Such factors include size of the project or responsibilities of different parties (for example, user requirements definition would not usually be the software developer's responsibility) or normal in-company practices.

Nevertheless, it remains true that a substantial first iteration of the project definition follows from imposition of the standards, particularly when it is considered that activity inputs and outputs (including deliverables) are prescribed in some detail.

Thus, properly applied, the standards could go some way to resolving the issue highlighted in the following quotation⁵: '... Performing individual software development tasks might well require a significant degree of creativity and imagination, but the definition and sequencing of those tasks can, and probably should, be predefined at an organisational level. In many organisations today, engineers spend too much of their creative energies reinventing (often dozens of times) processes.'

3. Some Central Schedule Considerations

3.1 Implications of the ESA standards

As a minimum, the ESA software engineering standards require that a software project incorporate the reviews and milestones depicted in Figure 3.1-1. (Similar requirements are stated in other standards - for example, p.14 of IEEE standard 1012¹⁵). For large projects, it might be necessary to include further reviews and milestones.

ACTIVITY	REVIEW	MILESTONE
Define user requirements (URD) ⇒	UR/R⇒	URD approved
Define software requirements (SRD)⇒	SR/R ⇒	SRD approved
Define architectural design (ADD) ⇒	AD/R⇒	ADD approved
Define detailed design (DDD) Prepare software user's manual (SUM) Do code Do unit tests Do integration tests Do system tests ⇒	DD/R⇒	DDD/SUM/Code approved
Install Do provisional acceptance tests ⇒		Provisional acceptance
Do final acceptance tests Do maintenance/operations Prepare project history doc. (PHD) ⇒		Final acceptance

Figure 3.1-1: Mandatory reviews & milestones in ESA SW Eng. standards

The following two points are highlighted for their impact on schedule:

- (a) Though not always an absolute requirement¹ it is desirable, ideally, that

¹ In fact, the ESA software engineering standards **do** require (pp 3-D2, 3-D5¹)

No.	Mandatory practice
UR11	The URD shall always be produced before a software project is started
D10	When the design of a major component is finished, a critical design review shall be convened to certify its readiness for implementation
D11	After production, the DD review (DD/R) shall consider the results of the verification activities and decide whether to transfer the software

each milestone be achieved before starting work on the succeeding phase.

(b) Each review entails a definite sequence of steps or stages (Fig. 3.1-2), for which an appreciable amount of time must be allocated.

1. Delivery (by authors) of review data to reviewers
2. Review period
3. Transmission of 'review item discrepancies' (RIDs) to authors
4. Review meeting
5. Implementation of review meeting decisions
6. Statement of milestone achievement

Figure 3.1-2: Sequence of review steps

As a very simple illustration of implications for schedule consider Table 3.1-1. Note that of the 6 week durations allowed in the table for review, 3 to 4 weeks would be taken up by 'Review Period', that is reviewers rather than authors would be active.

ITEM	SAMPLE DURATION (WEEKS)
SR/R PROCESS	6
AD/R PROCESS	6
DD/R PROCESS	6
TOTAL	18 [equals 35%, 17% and 9% of 1, 2 and 3 year project durations, resp.)

Table 3.1-1: Simple illustration of implications for schedule of formal review processes

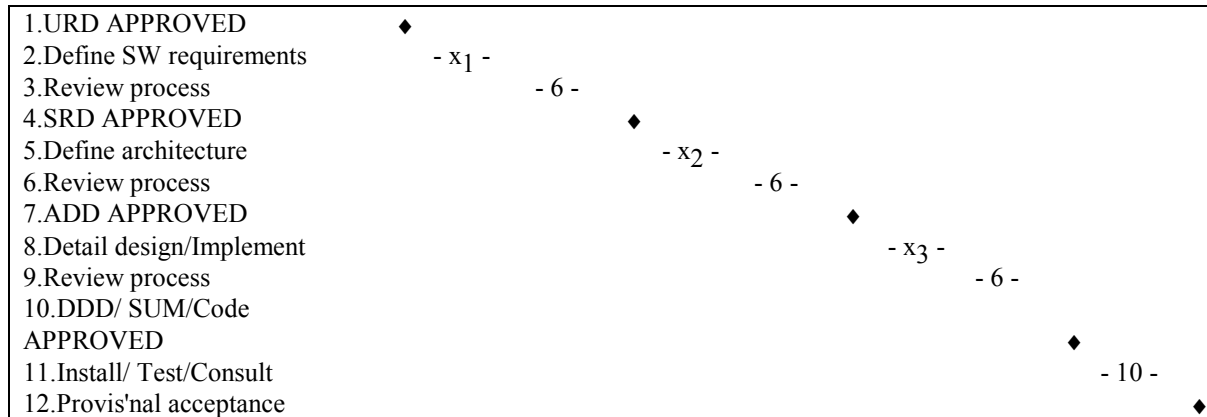
There is a legitimate pressure to reduce timescales in most projects and it sometimes (often?) happens that work proceeds on a phase before the milestone preceding it has been achieved. In particular, architectural design may be commenced before software requirements are baselined or coding done in advance of design approval.

A major disadvantage of phase overlapping is that the later phase is based on an informal level of information and will have to be revised when the information is formalised. This could entail a considerable configuration control as well as technical effort. A further point is that personnel from one phase should be among the reviewers of the work of a neighbouring phase - this will be difficult if there is phase overlap.

Further, if review processes are not given adequate time or resources, or are not properly conducted, there is a major risk that errors may remain undetected until later phases when their cost and schedule impact will be far greater.

In general, the cost, inefficiency and management overhead of attempting excessive parallel activities should be recognised. It should be appreciated that software (or other) projects require a certain 'natural' time to be properly mastered (p. 146⁶).

To clarify ideas, some simplistic schedule scenarios, based on ESA's mandatory reviews and milestones, are presented in Figure 3.1-2. Non-overlapping phases are assumed and the 'user requirements' and 'operations & maintenance' phases have been excluded (though, in practice, a software developer could well play a rôle in both).



(a) General Scheme

PROJECT DURATION (weeks)	53	80	106
'FIXED' ALLOCATION (REVIEWS + LEAD-UP TO PROVISIONAL ACCEPTANCE)	28	28	28
Define SW Requirements (x_1)	5	10	15
Define architecture (x_2)	5	10	15
Detail design / Implement (x_3)	15	32	48

(b) Possible Schedule Allocations

Figure 3.1-2: Some simplistic schedule scenarios based on ESA's mandatory reviews & milestones

3.2 Importance of choice of life cycle model

The illustrations of the previous section were confined to the 'waterfall' life cycle model. In fact, such a model may not be appropriate for a given project and attempts to enforce it could then lead to severe problems. It is to be expected, and is certainly confirmed in the author's experience that development of embedded software proceeds concurrently with the development of its 'target' system. This context imposes constraints not only on the software project's schedule but also on how it is specified, designed, implemented and tested.

The user requirements document (URD) is prepared, for such projects, by the 'embedding' system's developer. Because this system itself is under development the URD is subject to change. Similarly, related documents which define the interface between the software and other elements of the system are subject to change. However, the changes are likely to be systematic and largely predictable in character (though certainly not in their details).

In general, there is a progression from a minimal set of requirements to the complete definition. It is to be noted, also, that the system developer is likely to require

intermediate deliveries of the software as part of the *system* integration programme. Figure 3.2-1 depicts a general framework.

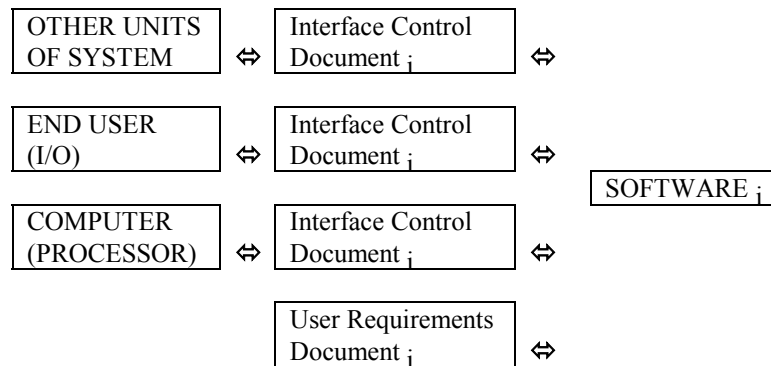


Figure 3.2-1(a): Evolution of inputs & SW - External interfaces

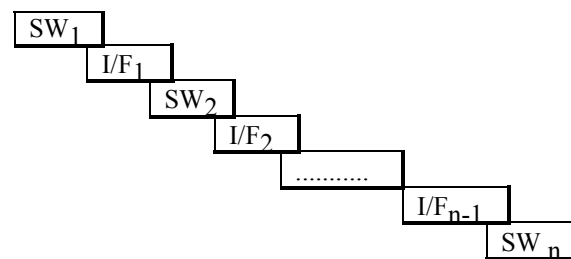


Figure 3.2-1(b): Evolution of inputs & SW - Internal interfaces

Notes:

(1) Double-headed arrows appear in part (a) of this figure to indicate that changes may be initiated by the software developer as well as by the system developer.

(ii) It should be noted that by 'SOFTWARE' in this figure is meant not just the code but all documentation (including test results). Thus, a complete development cycle is assumed for each version. If the amount of change is small from one version to the next then, of course, the development cycle will be correspondingly curtailed.

Figure 3.2-1 is a general framework that needs to be tailored to a specific project. Key objectives are

(a) Intermediate software deliveries are timely and adequate (but without superfluous features) to meet system integration needs

(b) There is a minimal (ideally none) amount of modification of previously delivered versions when a new delivery is made. Essentially, this means that a new version is comprised of the previous version (unaltered) plus additional features.

It is evident that successful achievement of these aims requires substantial joint planning and close cooperation between the system and software developers; contractual agreements should be drawn up with this in mind. At the same time, *both* must be conscious of the absolute importance of preserving formal interfaces and tight configuration control. This does not preclude informal, prototypical deliveries (for example, the system developer might need to perform rough performance checks or to

trouble-shoot another unit) but such deliveries must be clearly distinguished from the product software and separately planned for.

The scheme of Figure 3.2-1 may seem overly bureaucratic and rather difficult to manage. However, it should be realised that the alternative is incorporation of major changes in an unplanned manner. It is probable that this alternative would be costly and unpredictable, particularly in terms of testing effort.

Note on Maintenance/Evolution: The term 'maintenance' is something of a misnomer in software. Traditionally, for hardware, maintenance meant repairing something so that it returned to the condition it was in before a fault developed - this obviously does not make sense for software. However, if the scope of maintenance is extended to include the ease and economy with which system effectiveness can be maintained in the face of changing boundary conditions (p. 265⁶) then it becomes quite applicable for software. Strictly, this is evolution rather than maintenance (p. 24³) so that Figure 3.2-1 is again applicable. Maintenance also includes, of course, analysis and repair of design, coding and other errors. It is stressed that the cost of the maintenance phase strongly depends on how well the initial software development is performed (pp 24-28³).

3.3 Impact of errors detected during testing

It seems to be commonly accepted that (major) errors in software development are much more likely to be made in specifying requirements rather than during implementation (see Potter & Sinclair⁷, for example).

This reinforces the point made earlier that review processes (the basic error-detection mechanisms prior to testing) should be given a high priority in project schedules. This applies not only to formal reviews but also to reviews internal to the development team. Evidence in support of a policy of prioritizing reviews is provided by Grady¹⁷ who reports data from actual projects in which the 'average efficiency [defects found per hour] of code reading/inspections was 4.4 times better than other test techniques'.

The objective of software testing is to detect errors in the software. Therefore, tests can be considered successful if they do, in fact, uncover errors. However, a better situation would be that there are, in fact, no errors to be detected, that is that the software is error-free! This is because each time that an error is detected by test there is a significant schedule impact (see Table 3.3-1).

NORMAL TEST PROCEDURE	ADDITIONAL ACTIVITIES IF AN ERROR IS DETECTED
1. Execute test	1. Execute test
2. Analyse & report results	2. Analyse and report results
	3. Search for cause of the error
	4. Correct the error (code & documents)
	5. Execute test
	6. Analyse and report results
	7. If necessary, re-execute other tests

Table 3.3-1: Impact of error detection during test

Moreover, the overall schedule impact is difficult to predict for the following reasons:

- 1. Correction of one error may uncover another. The second error must be handled as the original one was. In principle, the same situation could arise for the second error, and so on.*
- 2. If part of the software has been changed to correct an error then retesting may be necessary to check that there are no side-effects introduced by the correction. Such retesting is definitely needed for software integration or higher-level tests. Decisions on the amount of retesting to be done are subjective and the retesting process is difficult to quantify.*
- 3. The total number of errors [which will be] uncovered during test is difficult to predict so that the likely overall schedule impact is hard to assess.*

Finally, the importance of uncovering errors as early as possible is stressed. This was already implied when noting the importance of review processes. Similarly, it is vital that each test phase is effective and that there is minimal 'leakage' of errors to higher level tests. For example, any mis-match of code to detailed design should be uncovered at unit test level and not persist to, say, system level where its detection, diagnosis and correction would be substantially more difficult, costly and time-consuming.

In essence, the aim should be to 'get it right first time'. The importance of this for each element of a complex, evolutionary development (such as in Figure 3.2-1) is self-evident.

4. Key Parameters of Software Projects

It was noted above that the ESA software engineering standards (or similar) form a basis for definition of a software project. This is because they prescribe a set of tasks, sequential links between tasks, an hierarchical work breakdown structure (to an extent), some scheduling constraints and (in a limited way) allocation of responsibilities.

It remains, of course, to tailor the standards to the needs of a particular project. It seems that this 'tailoring' can be automated to a certain extent through consideration of a limited number of factors and by suitable choice of certain parameters. For example,

1. LIFE CYCLE MODEL
2. CHOICE OF APPLICABLE PROCESSES
{For example, the user requirements definition phase might not apply}
3. PROJECT CATEGORY
{For example, the number of internal review activities might vary with criticality of the software}
4. SIZE
*{Properties of product: No. of design components, units, etc
Properties of process: No. of errors at different levels of V&T}*
5. ORGANISATION-SPECIFIC FACTORS
{For example, procedures for/personnel involved in unit test or internal reviews or design processes ...}
6. PROJECT CONSTRAINTS
{For example scheduling method (resource driven or fixed duration), date(s) for receipts of inputs, delivery schedule, budget}
7. PERSONNEL
{For example, work of experienced staff could be expected to require less monitoring (perhaps occasional detailed inspections rather than quasi-continuous ones)}

Figure 4-1: Project Tailoring: Possible factors and parameters

Remark: Specifically, regarding factor 3 (project category) comparison may be made with the approach in reference 20 for different software levels or to reference 23 for different safety integrity levels (SILs).

In principle, automation of much of this list is feasible and straightforward. In fact, as mentioned above, this has been done (in a prototype tool) by the author and is the means by which the results presented later in the paper were generated. Detailed aspects of the tool's specification and implementation are outside the scope of this paper.

This type of modeling tool may be used in various ways and at different stages in the life-time of a project. Perhaps one of its most important benefits is that it relieves users of the burden of detail and allows them instead to focus, in a dynamic way, on significant high-level project "behaviour".

It can be seen that there may be conflict between different factors of the above list. For example, an inappropriate choice of 'PROJECT CATEGORY' might lead to large numbers of errors at various test levels and thereby come in conflict with the delivery schedule. Similarly, it can certainly be expected that the allocated budget will be in conflict with some other factors! It is important that conflicts of these kinds be identified and resolved between all parties as early as possible. The resolution should be an agreed and feasible trade-off between conflicting factors². It is felt that the modelling approach adopted here will facilitate the reaching of agreement on effort and pricing estimates, in particular. In this context it is interesting to compare the foregoing list of 7 items with the 'underlying variables' [(1)users, (2)database, (3)personnel, (4)size, (5)language, (6)time, (7)project management environment,

² The essential issues are that agreement be reached on what is to be achieved in terms of, and on how to measure project time, cost, technical scope and quality.⁸

(8)complexity] postulated by Bergeron & St-Arnaud⁹ as being fundamental in assessing how well or otherwise various estimation methods perform.

This 'system' view of a software project - characterizing it in terms of a limited number of factors or parameters - is useful in helping to analyse project risk. It allows sensitivity analysis to be performed on various parameters which should enable critical items to be identified and corresponding risk-management measures¹⁰ to be taken.

5. Investigation of Model Projects

5.1 Approach

The purpose of this section is to define a representative set of model projects to illustrate the previous discussion, and to highlight and to make an initial attempt to quantify particular aspects.

For clarity, the number of illustrations is deliberately quite limited. The aim is to focus on key ideas rather than to provide complete coverage.

A particular restriction is to consider just two types of labour, USER (software user or acquirer) and DEVP (software developer). Of course, it could be very useful to break DEVP down further, for example into project manager, analyst, designer, tester, quality engineer, and so on. However, this is not done here, in the interest of avoiding excessive detail. For model 6 (evolutionary model) four 'teams' of software developers are considered.

5.2 Selection of model projects

The seven parameters or factors listed in section 4 are used as the basis for defining the set of model projects. The choices considered for these parameters and factors are listed in Table 5.2-1.

	CHOICE 1	CHOICE 2
LIFE CYCLE MODEL	<u>Waterfall</u>	<u>Evolutionary</u> (See model 6)
CHOICE OF APPLICABLE PROCESSES	<u>Full</u> (All processes though some not specified in detail)	<u>Reduced</u> (Omission of separate SPM, SQA and SCM (Part 2 ¹) processes + post DD/R) activities and acceptance tests.
PROJECT CATEGORY	<u>Class A</u> (High level of criticality- full programme of internal reviews)	<u>Class B</u> (Reduced level of criticality- internal reviews only before formal deliveries) [For SW requirements, architecture and unit production, reduction is to a review of just 1 lowest level item]
SIZE-PRODUCT ³	<u>Large</u> (S _B , S _D , A _B , A _D , N _U) = (4, 3, 4, 3, 16)	<u>Medium</u> (S _B , S _D , A _B , A _D , N _U) = (4, 2, 4, 2, 4)
SIZE-PROCESS ⁴	<u>Big</u> (E _I , E _I , E _S) = (4, 20, 20)	<u>Zero</u> (E _I , E _I , E _S) = (0, 0, 0)
ORGANISATION-SPECIFIC FACTORS	- (see section 6 for effort & duration allocated to individual activity elements)	-
PROJECT CONSTRAINTS ⁵	OUTPUT <u>Fixed-duration</u>	OUTPUT <u>Resource-driven</u> (Imposed by 'levelling' using Microsoft Project ⁴ - see section 6)
PERSONNEL	- (Choices for other parameters may be interpreted as implying choices of this item)	-

Table 5.2-1: Possible Choices (restricted) for Parameters & Factors

The selected project models, defined in terms of these parameter and factor choices, are presented in Table 5.2-2.

³ For S_B, S_D, A_B, A_D and N_U see section A.6.

For clarity, though at the risk of some over-simplification, the number of (each of) designs, case specifications, procedures and reports for each of integration, system and acceptance tests (Chapter 4 (Part 2)¹) is taken to be 1, that is, all individual elements are assumed to be grouped and documented together.

⁴ For E_U, E_I and E_S see sections A.2, A.3 and A.4, respectively.

⁵ The qualifier **OUTPUT** means that the parameter or factor is varied during post-processing so that results may be reported (within the same basic model) for all possible choices of the item.

MODEL PROJECT	1	2	3	4	5	6
LIFE CYCLE MODEL	Waterfall	Waterfall	Waterfall	Waterfall	Waterfall	Evolutionary
CHOICE OF APPLICABLE PROCESSES	Full	Reduced	Reduced	Reduced	Reduced	Reduced + SVVP/AT, STD, Installation - to provisional acceptance
PROJECT CATEGORY	Class A	Class A	Class A	Class B	Class B	Class A
SIZE-PRODUCT	Medium	Medium	Large	Large	Large	$(S_B, S_D, A_B, A_D, N_U) =$ (2, 2, 2, 2, 2) for Element 1 (3, 2, 3, 2, 3) for Element 2 (4, 2, 4, 2, 4) for Element 3 (4, 2, 4, 2, 4) for Element 4
SIZE-PROCESS	Zero	Zero	Zero	Zero	Big	Zero
ORGANISATION-SPECIFIC FACTORS	-	-	-	-	-	-
PROJECT CONSTRAINTS	-	Fixed-Duration & Resource-Driven	Fixed-Duration & Resource-Driven	Fixed-Duration & Resource-Driven	Fixed-Duration & Resource-Driven	Fixed-Duration & Resource-Driven
PERSONNEL	-	-	-	-	-	-

Table 5.2-2: Definition of Model Projects

Model 1 is included to provide an overview of the several processes active during a software development project. It includes some processes which are omitted in subsequent models. Data on durations and resource allocations are not reported for this model.

Models 2 and 3 are to be compared for insight into the effect of product size.

Models 3 and 4 are to be compared for insight into the effect of project category.

Models 4 and 5 are to be compared for insight into the impact on effort and duration of errors detected and repaired during test.

Finally, model 6 is included as representative of an evolutionary rather than waterfall life-cycle model.

6. Presentation of Results

6.1 Model 1

Figure 6.1-1 is a task sheet showing the several activities of which the model is comprised, and the precedence relationships between them. For brevity, details of some activities have been suppressed.

ID	Name	Preds	ID	Name	Preds
1	Development platform		69	... [NOT modelled in detail here]	
2	Select a programming language(s)	6	70	Software configuration management	
3	Milestones		71	Configuration identification	4
4	SW kickoff		72	Configuration item storage	4
5	URD approved	17	73	Configuration change control	
6	SRD approved	23	74	Decide on change control levels	4
7	ADD approved	29	75	Establish change procedures	4
8	Code/DDD/SUM approved	35		Establish problem reporting procedures	4

ID	Name	Preds	ID	Name	Preds
9	Provisional acceptance - transfer done	38,147,144	76	Configuration status accounting	4
10	Final acceptance - maintenance done	42,153	77	Configure releases	4
11	External major reviews		78	Configuration management plan	
12	UR/R			...	
13	Circulate for external review	125, 68, 4, 52, 81,108	91	Software quality assurance	
14	Prepare (external review) RIDs	13	92	Verify ... both planned and actual	
15	Prepare answers to RIDs (external rev.	14	93	Verify Management (QA)	4
16	Hold meeting (external review)	15	94	Verify Documentation (QA)	4
17	Update after meeting (external review)	16	95	Verify St'rds/Pract's/Conv'ns (QA)	4
18	SR/R		96	Verify Review mechanisms (QA)	4
19	Circulate for external review	130, 5, 55, 84,111	97	Verify Testing activities (QA)	4
20	Prepare (external review) RIDs	19	98	Verify PRACAS (QA)	4
21	Prepare answers to RIDs (external rev.	20	99	Verify Tools/Te'ques/Methods (QA)	4
22	Hold meeting (external review)	21	100	Verify Code and media control (QA)	4
23	Update after meeting (external review)	22	101	Verify Supplier control (QA)	4
24	AD/R		102	Verify Records col./main./retent.(QA)	4
25	Circulate for external review	2,135, 6, 58, 87,114	103	Verify Training (QA)	4
26	Prepare (external review) RIDs	25	104	Verify Risk management (QA)	4
27	Prepare answers to RIDs (external rev.	26	105	Quality assurance plan	
28	Hold meeting (external review)	27		...	
29	Update after meeting (external review)	28	118	Standards	
30	DD/R		119	Coding standards	
31	Circulate for external review	7,141,138, 61, 64, 90, 117	120	Prepare Coding standards	7
32	Prepare (external review) RIDs	31	121	Execute internal review (general)	120
33	Prepare answers to RIDs (external rev.	32	122	Define user requirements	
34	Hold meeting (external review)	33	123	Capture user requirements	4
35	Update after meeting (external review)	34	124	Determine operational environment	123
36	Provisional acceptance		125	Specify user requirements	124
37	Convene SRB (for prov. acceptance)	8	126	Define SW requirements	
38	Make recommendations re prov. accept.	37	127	Define SW reqts1	
39	Final acceptance		128	Construct a logical model	5
40	Convene SRB (for final acceptance)	150	129	Specify software requirements	128
41	Consult parties to final acceptance	150	130	Execute internal review (general)	129
42	Reach decision on final acceptance	40, 41	131	Define SW architecture	
43	Software project management		132	Define SW arch.1	
44	Organise (Proj. Man. task)	4	133	Construct a physical model	6
45	Lead (Proj. Man. task)	4	134	Specify architectural design	133
46	Manage risk (Proj. Man. task)	4	135	Execute internal review (general)	134
47	Technically manage (Proj. Man. task)	4	136	Software user manual	
48	Plan-Schedule-Budget (Proj. Man. task)	4	137	Prepare SUM	7
49	Management plan		138	Execute internal review (general)	137
50	SPMP - SR Phase		139	Produce software units	
51	Prepare SPMP/SR	4	140	Design/code/test SW unit 1	
52	Execute internal review (general)	51	141	Produce the unit	121, 7
53	SPMP - AD Phase		142	Installation	
54	Prepare SPMP/AD	5	143	Check deliverables against checklist	8
55	Execute internal review (general)	54	144	Build executable for target machine	143
56	SPMP - DD Phase(a)		145	Software transfer document	
57	Prepare SPMP/DD(initial)	6	146	Prepare STD	8
58	Execute internal review (general)	57	147	Execute internal review (general)	146
59	SPMP - DD Phase(b)		148	Warranty period	
60	Prepare SPMP/DD(update)	7	149	Correct errors (during warranty)	9
61	Execute internal review (general)	60	150	Document corrections (during warranty)	149
62	SPMP - TR Phase		151	Project history document	
63	Prepare SPMP/TR	7	152	Prepare PHD	9
64	Execute internal review (general)	63	153	Execute internal review (general)	152
65	Software verification and validation				

Figure 6.1-1: Model 1 - Task sheet

6.2 Model 2

Figures 6.2-1 and 6.2-1(a) show the expenditure (in hours) of the software developer resource Devp as a function of time in quarters and years, respectively. No attempt to level allocation of resource has been made for these figures.

Figure 6.2-2(quarters) and Figure 6.2-2(a)(years) show the expenditure profile of resource Devp following attempted levelling to accommodate a maximum availability of 2 units of resource Devp. Similarly, Figures 6.2-3 and 6.2-3(a) show the expenditure profile of Devp in quarters and years, respectively, following levelling to accommodate a maximum availability of 4 units of resource Devp.

Figure 6.2-4 lists the several activities of which the model is comprised showing durations (prior to any resource levelling exercise), precedence relationships and resource allocations. Some activities are given at summary level only (indicated by gaps in ID numbers) but sufficient structure is shown to allow most omitted detail to be inferred. Note that the duration of 'Milestones' activity is equal to the complete project duration.

Summary statistics for the model are,

Total effort of Devp (hours)	11573
Duration (before levelling)	536(working) days or 2.0 years(approx)
Duration (levelling on 2 units of Devp)	4.0 years(approx)
Duration (levelling on 4 units of Devp)	2.3 years(approx)

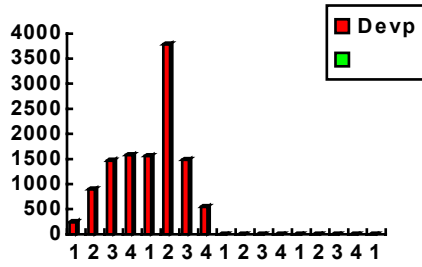


Figure 6.2-1: Model 2 - Expenditure of Devp (before levelling) (qrts)

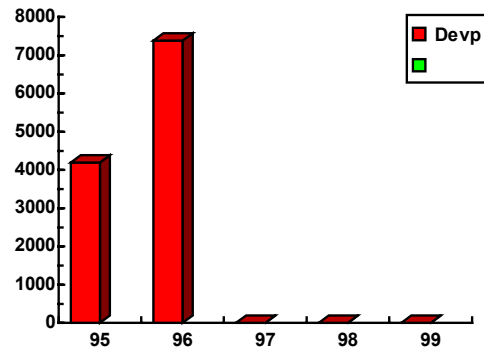


Figure 6.2-1(a): Model 2 - Expenditure of Devp (before levelling) (years)

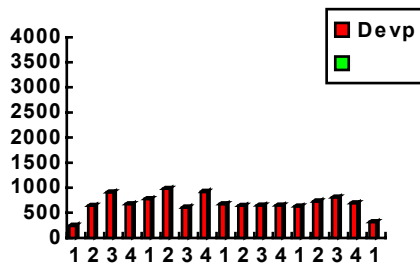


Figure 6.2-2: Model 2 - Expenditure of Devp (levelling at 2 units) (qrts)

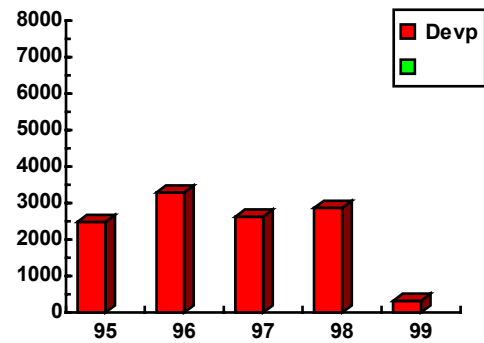


Figure 6.2-2(a): Model 2 - Expenditure of Devp (levelling at 2 units) (years)

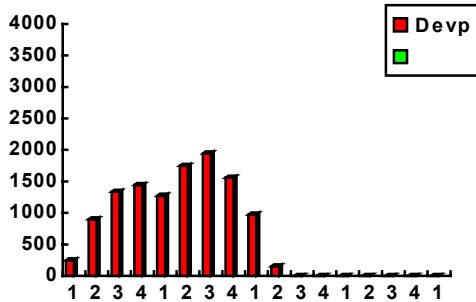


Figure 6.2-3: Model 2 - Expenditure of Devp (levelling at 4 units) (qrts)

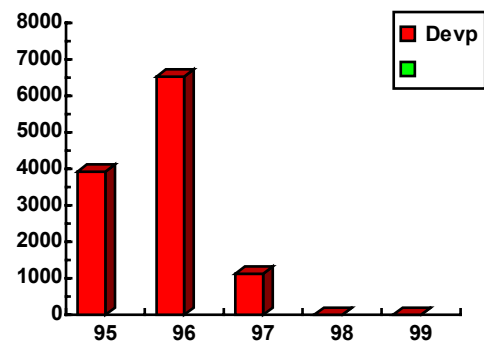


Figure 6.2-3(a): Model 2 - Expenditure of Devp (levelling at 4 units) (years)

ID	Name	Dur.	Preds	Resource Names
1	Milestones	536d		
2	SW kickoff	0d		User[0.00],Devp[0.00]
3	URD approved	0d	13	User[0.00],Devp[0.00]
4	SRD approved	0d	19	User[0.00],Devp[0.00]
5	ADD approved	0d	25	User[0.00],Devp[0.00]
6	Code/DDD/SUM approved	0d	31	User[0.00],Devp[0.00]
7	External major reviews	456d		
8	UR/R	26d		
9	Circulate for external review	1d	97,35,2	User[0.00],Devp[1.00]
10	Prepare (external review) RIDs	15d	9	User[2.00],Devp[0.00]
11	Prepare answers to RIDs (external rev.	3d	10	User[0.00],Devp[3.00]
12	Hold meeting (external review)	2d	11	User[2.00],Devp[4.00]
13	Update after meeting (external review)	5d	12	User[0.00],Devp[1.40]
14	SR/R	26d		
20	AD/R	26d		
26	DD/R	26d		
32	Software verification and validation	510d		
33	SVVP - SR Phase	29d		
34	Prepare SVVP/SR	20d	2	User[0.00],Devp[1.00]
35	Execute internal review (general)	9d	34	User[0.00],Devp[1.30]
36	SVVP - AD Phase	29d		
39	SVVP - DD Phase	29d		
42	SVVP - System tests	404d		
43	SVVP/ST(Plan)	29d		
44	Prepare SVVP/ST(Plan)	20d	3	User[0.00],Devp[1.00]
45	Execute internal review (general)	9d	44	User[0.00],Devp[1.30]
46	SVVP/ST(Designs)	59d		
47	Prepare/Review test design 1	59d		
51	SVVP/ST(Specs)	34d		
52	Prepare/Review test spec 1	34d		
56	SVVP/ST(Procedures)	59d		
57	Prepare/Review test procedure 1	59d		
61	SVVP/ST(Reports)	22d		
62	Execute/Report test 1	22d		
66	SVVP - Integration tests	278d		
90	SVVP - Unit tests	29d		
91	SVVP/UT(Plan)	29d		
92	Prepare SVVP/UT(Plan)	20d	5	User[0.00],Devp[1.00]
93	Execute internal review (general)	9d	92	User[0.00],Devp[1.30]
94	Define user requirements	80d		
95	Capture user requirements	30d	2	User[2.00],Devp[0.00]
96	Determine operational environment	30d	95	User[2.00],Devp[1.10]
97	Specify user requirements	20d	96	User[1.00],Devp[0.00]
98	Define SW requirements	78d		
99	Define SW reqts1	78d		
100	Construct a logical model	20d	3	User[0.00],Devp[1.00]
101	Specify software requirements	10d	100	User[0.00],Devp[1.00]
102	Execute internal review (general)	9d	101	User[0.00],Devp[1.30]
103	Define SW reqts11	39d		
107	Define SW reqts12	39d		
111	Define SW reqts13	39d		
115	Define SW reqts14	39d		
119	Define SW architecture	78d		
120	Define SW arch.1	78d		
121	Construct a physical model	20d	4	User[0.00],Devp[1.00]
122	Specify architectural design	10d	121	User[0.00],Devp[1.00]
123	Execute internal review (general)	9d	122	User[0.00],Devp[1.30]
124	Define SW arch.11	39d		
125	Construct a physical model	20d	123	User[0.00],Devp[1.00]
126	Specify architectural design	10d	125	User[0.00],Devp[1.00]
127	Execute internal review (general)	9d	126	User[0.00],Devp[1.30]
128	Define SW arch.12	39d		
132	Define SW arch.13	39d		
136	Define SW arch.14	39d		
140	Software user manual	39d		
141	Prepare SUM	30d	5	User[0.00],Devp[1.00]
142	Execute internal review (general)	9d	141	User[0.00],Devp[1.30]
143	Produce software units	69.5d		
144	Design/code/test SW unit 1	69.5d		
145	Produce the unit	69.5d	5, 93	User[0.00],Devp[1.25]
146	Design/code/test SW unit 2	69.5d		
148	Design/code/test SW unit 3	69.5d		
150	Design/code/test SW unit 4	69.5d		

Figure 6.2-4: Model 2 - Task sheet (durations before levelling)

6.3 Model 3

Figure 6.3-1 shows the expenditure profile of the software developer resource Devp. No attempt to level allocation of resource has been made (for this figure).

Figure 6.3-2 shows the expenditure profile of resource Devp following levelling to accommodate a maximum availability of 8 units of resource Devp.

Figure 6.3-3 lists the several activities of which the model is comprised showing durations (prior to any resource levelling exercise), precedence relationships and

resource allocations. Several activities are given at summary level only but sufficient structure is shown to allow most omitted detail to be inferred (Figure 6.2-4 may be referred to for some details). Note that the duration of 'Milestones' activity is equal to the complete project duration.

In preparation of this paper, two Gantt charts were prepared - though not reproduced here - that showed, respectively, the model project's schedule prior to any resource levelling exercise and following levelling at 8 units of Devp.

From the second of these (case of levelling at 8 units of Devp) the extent of different project phases was observed to be

1995	1996	1997	1998	1999
UR&SR	AD	Production		IT&ST

This, together with Figure 6.3-2, allows a comparison to be made with some generalized effort distributions presented by Ratcliff¹⁹ (pp281, 282), excluding the Operations & Maintenance phase. As could be anticipated, the closest match to Figure 6.3-2 is that of a project 'executed with sound software engineering techniques'.

Summary statistics for the model are,

Total effort of Devp (hours)	45550
Duration (before levelling)	961(working) days or 3.5 years(approx)
Duration (levelling on 8 units of Devp)	4.7 years(approx)

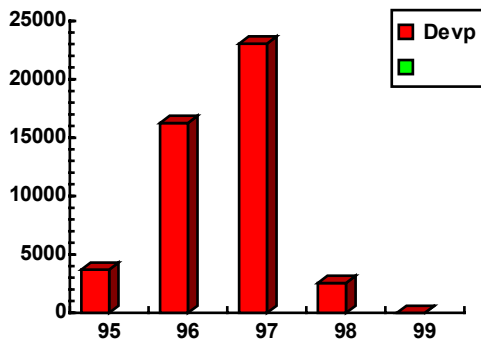


Figure 6.3-1: Model 3 - Expenditure of Devp (before levelling) (years)

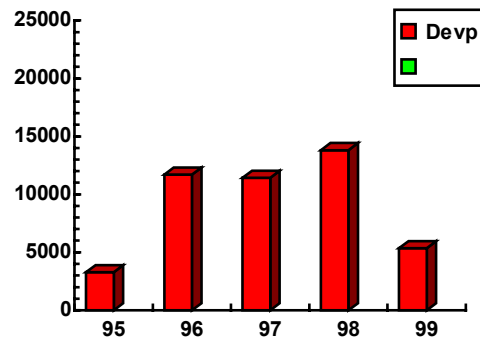


Figure 6.3-2: Model 3 - Expenditure of Devp (levelling at 8 units) (years)

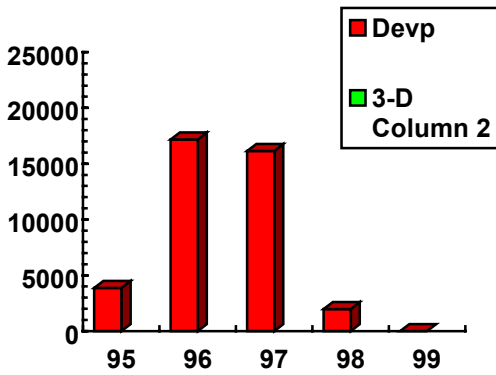


Figure 6.4-1: Model 4 - Expenditure of Devp (before levelling) (years)

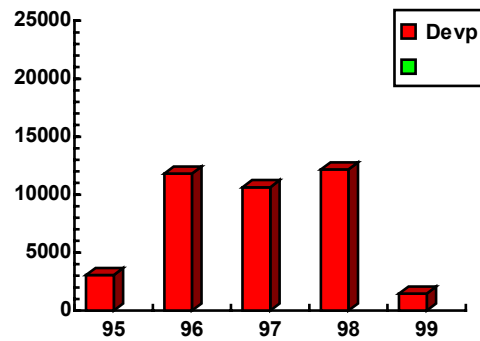


Figure 6.4-2: Model 4 - Expenditure of Devp (levelling at 8 units) (years)

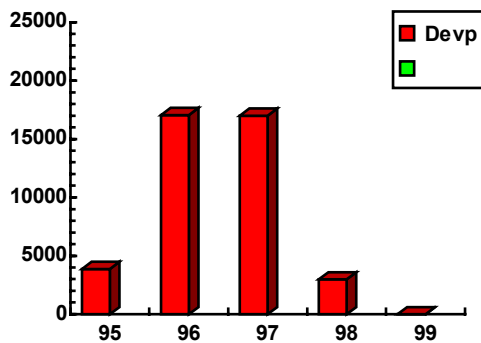


Figure 6.5-1: Model 5 - Expenditure of Devp (before levelling) (years)

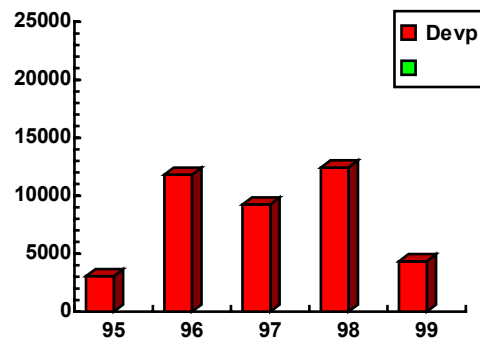


Figure 6.5-2: Model 5 - Expenditure of Devp (levelling at 8 units) (years)

ID	Name	Dur.	Preds	Resource Names
1	Milestones	961d		
2	SW kickoff	0d		User[0.00],Devp[0.00]
3	URD approved	0d	13	User[0.00],Devp[0.00]
4	SRD approved	0d	19	User[0.00],Devp[0.00]
5	ADD approved	0d	25	User[0.00],Devp[0.00]
6	Code/DDD/SUM approved	0d	31	User[0.00],Devp[0.00]
7	External major reviews	761d		
8	UR/R	32d		
9	Circulate for external review	1d	97,35,2	User[0.00],Devp[1.00]
10	Prepare (external review) RIDs	15d	9	User[4.00],Devp[0.00]
11	Prepare answers to RIDs (external rev.)	3d	10	User[0.00],Devp[6.00]
12	Hold meeting (external review)	3d	11	User[3.00],Devp[4.00]
13	Update after meeting (external review)	10d	12	User[0.00],Devp[3.00]
14	SR/R	32d		
20	AD/R	32d		
26	DD/R	32d		
32	Software verification and validation	929d		
94	Define user requirements	200d		
98	Define SW requirements	117d		
99	Define SW reqts1	117d		
100	Construct a logical model	20d	3	User[0.00],Devp[1.00]
101	Specify software requirements	10d	100	User[0.00],Devp[1.00]
102	Execute internal review (general)	9d	101	User[0.00],Devp[1.30]
103	Define SW reqts11	78d		
104	Construct a logical model	20d	102	User[0.00],Devp[1.00]
105	Specify software requirements	10d	104	User[0.00],Devp[1.00]
106	Execute internal review (general)	9d	105	User[0.00],Devp[1.30]
107	Define SW reqts111	39d		
108	Construct a logical model	20d	106	User[0.00],Devp[1.00]
109	Specify software requirements	10d	108	User[0.00],Devp[1.00]
110	Execute internal review (general)	9d	109	User[0.00],Devp[1.30]
111	Define SW reqts112	39d		
115	Define SW reqts113	39d		
119	Define SW reqts114	39d		
123	Define SW reqts12	78d		
143	Define SW reqts13	78d		
163	Define SW reqts14	78d		
183	Define SW architecture	117d		
184	Define SW arch.1	117d		
185	Construct a physical model	20d	4	User[0.00],Devp[1.00]
186	Specify architectural design	10d	185	User[0.00],Devp[1.00]
187	Execute internal review (general)	9d	186	User[0.00],Devp[1.30]
188	Define SW arch.11	78d		
189	Construct a physical model	20d	187	User[0.00],Devp[1.00]
190	Specify architectural design	10d	189	User[0.00],Devp[1.00]
191	Execute internal review (general)	9d	190	User[0.00],Devp[1.30]
192	Define SW arch.111	39d		
196	Define SW arch.112	39d		
200	Define SW arch.113	39d		
204	Define SW arch.114	39d		
208	Define SW arch.12	78d		
228	Define SW arch.13	78d		
248	Define SW arch.14	78d		
268	Software user manual	69d		
271	Produce software units	69.5d		
272	Design/code/test SW unit 1	69.5d		
274	Design/code/test SW unit 2	69.5d		
276	Design/code/test SW unit 3	69.5d		
278	Design/code/test SW unit 4	69.5d		
280	Design/code/test SW unit 5	69.5d		
282	Design/code/test SW unit 6	69.5d		
284	Design/code/test SW unit 7	69.5d		
286	Design/code/test SW unit 8	69.5d		
288	Design/code/test SW unit 9	69.5d		
290	Design/code/test SW unit10	69.5d		
292	Design/code/test SW unit11	69.5d		
294	Design/code/test SW unit12	69.5d		
296	Design/code/test SW unit13	69.5d		
298	Design/code/test SW unit14	69.5d		
300	Design/code/test SW unit15	69.5d		
302	Design/code/test SW unit16	69.5d		

Figure 6.3-3: Model 3 - Task sheet (durations before levelling)

6.4 Model 4

Figure 6.4-1 shows the expenditure profile of the software developer resource Devp. No attempt to level allocation of resource has been made (for this figure).

Figure 6.4-2 shows the expenditure profile of resource Devp following levelling to accommodate a maximum availability of 8 units of resource Devp.

Figure 6.4-3 lists the several activities of which the model is comprised showing durations (prior to any resource levelling exercise), precedence relationships and resource allocations. Several activities are given at summary level only but sufficient structure is shown to allow most omitted detail to be inferred (Figure 6.2-4 may be referred to for some details). Note that the duration of 'Milestones' activity is equal to the complete project duration. Particular attention is drawn to representative dummy activities 102, 166, 187, 251, 255, 259, and 263; such dummy activities are one of the mechanisms used to model a project of class B. Similarly, note the curtailed duration of activities 274, 276, ..., 300 in comparison with activity 302 (see section A.1).

Summary statistics for the model are,

Total effort of Devp (hours)		39130
Duration (before levelling)	925 (working) days or 3.4 years(approx)	
Duration (levelling on 8 units of Devp)		4.3 years(approx)

6.5 Model 5

Figure 6.5-1 shows the expenditure profile of the software developer resource Devp. No attempt to level allocation of resource has been made (for this figure).

Figure 6.5-2 shows the expenditure profile of resource Devp following levelling to accommodate a maximum availability of 8 units of resource Devp.

Figure 6.5-3 lists the several activities of which the model is comprised showing durations (prior to any resource levelling exercise), precedence relationships and resource allocations. Several activities are given at summary level only but sufficient structure is shown to allow most omitted detail to be inferred (Figure 6.2-4 may be referred to for some details). Note that the duration of 'Milestones' activity is equal to the complete project duration. Particular attention is drawn to activities 66-67, 92-93 and 308-...324 as these are the means by which error detection during test is incorporated in the model.

Summary statistics for the model are,

Total effort of Devp (hours)		40888
Duration (before levelling)	1043 (working) days or 3.8 years(approx)	
Duration (levelling on 8 units of Devp)		4.8 years(approx)

ID	Name	Dur.	Pred s	Resource Names
1	Milestones	925d		
7	External major reviews	725d		
32	Software verification and validation	893d		
94	Define user requirements	200d		
95	Capture user requirements	120d	2	User[2.00],Devp[0.00]
96	Determine operational environment	40d	95	User[3.00],Devp[2.00]
97	Specify user requirements	40d	96	User[2.00],Devp[0.00]
98	Define SW requirements	99d		
99	Define SW reqts1	99d		
100	Construct a logical model	20d	3	User[0.00],Devp[1.00]
101	Specify software requirements	10d	100	User[0.00],Devp[1.00]
102	Dummy (SW reqts)	0d	101	User[0.00],Devp[0.00]
103	Define SW reqts11	60d		
123	Define SW reqts12	60d		
143	Define SW reqts13	60d		
163	Define SW reqts14	69d		
164	Construct a logical model	20d	102	User[0.00],Devp[1.00]
165	Specify software requirements	10d	164	User[0.00],Devp[1.00]
166	Dummy (SW reqts)	0d	165	User[0.00],Devp[0.00]
167	Define SW reqts141	30d		
171	Define SW reqts142	30d		
175	Define SW reqts143	30d		
179	Define SW reqts144	39d		
180	Construct a logical model	20d	166	User[0.00],Devp[1.00]
181	Specify software requirements	10d	180	User[0.00],Devp[1.00]
182	Execute internal review (general)	9d	181	User[0.00],Devp[1.30]
183	Define SW architecture	99d		
184	Define SW arch.1	99d		
185	Construct a physical model	20d	4	User[0.00],Devp[1.00]
186	Specify architectural design	10d	185	User[0.00],Devp[1.00]
187	Dummy (Arch)	0d	186	User[0.00],Devp[0.00]
188	Define SW arch.11	60d		
208	Define SW arch.12	60d		
228	Define SW arch.13	60d		
248	Define SW arch.14	69d		
249	Construct a physical model	20d	187	User[0.00],Devp[1.00]
250	Specify architectural design	10d	249	User[0.00],Devp[1.00]
251	Dummy (Arch)	0d	250	User[0.00],Devp[0.00]
252	Define SW arch.141	30d		
253	Construct a physical model	20d	251	User[0.00],Devp[1.00]
254	Specify architectural design	10d	253	User[0.00],Devp[1.00]
255	Dummy (Arch)	0d	254	User[0.00],Devp[0.00]
256	Define SW arch.142	30d		
257	Construct a physical model	20d	251	User[0.00],Devp[1.00]
258	Specify architectural design	10d	257	User[0.00],Devp[1.00]
259	Dummy (Arch)	0d	258	User[0.00],Devp[0.00]
260	Define SW arch.143	30d		
261	Construct a physical model	20d	251	User[0.00],Devp[1.00]
262	Specify architectural design	10d	261	User[0.00],Devp[1.00]
263	Dummy (Arch)	0d	262	User[0.00],Devp[0.00]
264	Define SW arch.144	39d		
265	Construct a physical model	20d	251	User[0.00],Devp[1.00]
266	Specify architectural design	10d	265	User[0.00],Devp[1.00]
267	Execute internal review (general)	9d	266	User[0.00],Devp[1.30]
268	Software user manual	69d		
271	Produce software units	69.5d		
272	Design/code/test SW unit 1	50d		
273	Produce the unit	50d	5,93	User[0.00],Devp[1.22]
274	Design/code/test SW unit 2	50d		
276	Design/code/test SW unit 3	50d		
278	Design/code/test SW unit 4	50d		
280	Design/code/test SW unit 5	50d		
282	Design/code/test SW unit 6	50d		
284	Design/code/test SW unit 7	50d		
286	Design/code/test SW unit 8	50d		

ID	Name	Dur.	Preds	Resource Names
288	Design/code/test SW unit 9	50d		
290	Design/code/test SW unit10	50d		
292	Design/code/test SW unit11	50d		
294	Design/code/test SW unit12	50d		
296	Design/code/test SW unit13	50d		
298	Design/code/test SW unit14	50d		
300	Design/code/test SW unit15	50d		
302	Design/code/test SW unit16	69.5d		

Figure 6.4-3: Model 4 - Task sheet (durations before levelling)

ID	Name	Dur.	Preds	Resource Names
1	Milestones	1043.3d		
7	External major reviews	843.3d		
32	Software verification and validation	1011.3d		
33	SVVP - SR Phase	39d		
36	SVVP - AD Phase	39d		
39	SVVP - DD Phase	39d		
42	SVVP - System tests	779.3d		
43	SVVP/ST(Plan)	39d		
46	SVVP/ST(Designs)	109d		
51	SVVP/ST(Specs)	59d		
56	SVVP/ST(Procedures)	109d		
61	SVVP/ST(Reports)	135.4d		
62	Execute/Report test 1	61d		
63	Execute test	40d	60,93	User[0.00],Devp[2.00]
64	Prepare report	12d	63	User[0.00],Devp[2.00]
65	Execute internal review (general)	9d	64	User[0.00],Devp[6.00]
66	Handle system test time errors	74.4d		
67	System test repeats	74.4d	65	User[0.00],Devp[1.20]
68	SVVP - Integration tests	512.9d		
69	SVVP/IT(Plan)	39d		
72	SVVP/IT(Designs)	109d		
77	SVVP/IT(Specs)	59d		
82	SVVP/IT(Procedures)	109d		
87	SVVP/IT(Reports)	104.9d		
88	Execute/Report test 1	61d		
89	Execute test	40d	86,309-324	User[0.00],Devp[2.00]
90	Prepare report	12d	89	User[0.00],Devp[2.00]
91	Execute internal review (general)	9d	90	User[0.00],Devp[6.00]
92	Handle int test time errors	43.9d		
93	Integ test repeats	43.9d	91	User[0.00],Devp[1.02]
94	SVVP - Unit tests	39d		
98	Define user requirements	200d		
102	Define SW requirements	99d		
187	Define SW architecture	99d		
272	Software user manual	69d		
275	Produce software units	76.4d		
276	Design/code/test SW unit 1	50d		
277	Produce the unit	50d	5,97	User[0.00],Devp[1.22]
278	Design/code/test SW unit 2	50d		
280	Design/code/test SW unit 3	50d		
282	Design/code/test SW unit 4	50d		
284	Design/code/test SW unit 5	50d		
286	Design/code/test SW unit 6	50d		
288	Design/code/test SW unit 7	50d		
290	Design/code/test SW unit 8	50d		
292	Design/code/test SW unit 9	50d		
294	Design/code/test SW unit10	50d		
296	Design/code/test SW unit11	50d		
298	Design/code/test SW unit12	50d		
300	Design/code/test SW unit13	50d		
302	Design/code/test SW unit14	50d		
304	Design/code/test SW unit15	50d		
306	Design/code/test SW unit16	69.5d		
308	Handle unit test time errors	6.9d		
309	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
310	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
311	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
312	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
313	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
314	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
315	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
316	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
317	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]
318	Unit test repeats	6.9d	277,279 ..	User[0.00],Devp[1.04]

ID	Name	Dur.	Preds	Resource Names
319	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]
320	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]
321	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]
322	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]
323	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]
324	Unit test repeats	6.9d	277,279	User[0.00],Devp[1.04]

Figure 6.5-3: Model 5 - Task sheet (durations before levelling)

6.6 Model 6

Figure 6.6-1 shows the expenditure profile of the software acquirer/user resource User and of each of the software developer resources Team1, Team2, Team3 and Team4. Figure 6.6-2 shows the corresponding overall expenditure of development resources (Team1 to Team4).

For each of Figures 6.6-1 and 6.6-2, (A) corresponds to the case where no levelling of resource allocations has been performed (fixed duration), (B) corresponds to where resources have been levelled to accommodate a maximum availability of 4 units of each resource type (each element is roughly commensurate with model 2), and (C) differs from (B) in having a maximum availability of 6 units of resource Team1 but with the constraint that the task 'APPLY 1 at system level' must start no later than October 10th, 1997.

Figure 6.6-3 is a summarising task sheet which depicts the high level structure of the four sub-projects, showing durations (prior to levelling), precedence relationships and resource allocations. Omitted detail may be largely inferred from earlier models.

Summary statistics for the model are,

Total effort of User (hours)	11790
Total effort of Team1 (hours)	14119
Total effort of Team2 (hours)	14946
Total effort of Team3 (hours)	15548
Total effort of Team4 (hours)	14873
Total devp effort (Team1 to Team4)	59486
Devp effort (Elements 1 to 4 only)*	51161
Duration (before levelling)	6.7 years(approx)
Duration (Levelling B)	8.0 years(approx)
Duration (Levelling C)	8.0 years(approx)

* Less activities 1, 155, 319, 320, 494, 495, 669, 670 (Figure 6.6-3)

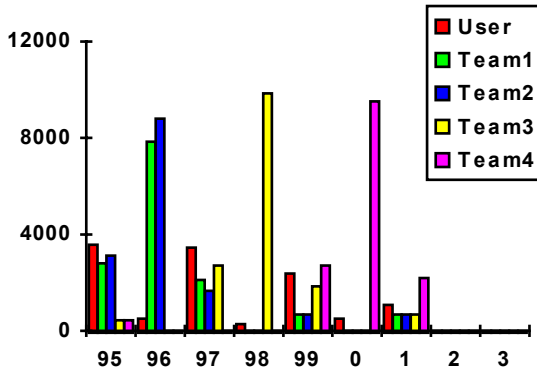


Figure 6.6-1(A): Model 6 - Expenditure of all 5 resource categories (before levelling)

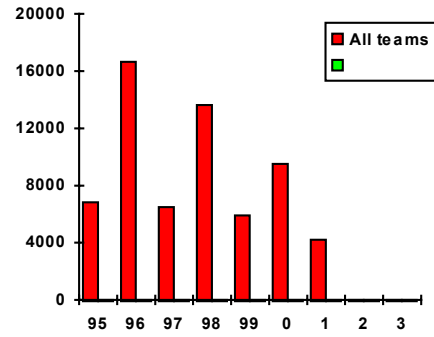


Figure 6.6-2(A): Model 6 - Overall development resource expenditure (before levelling)

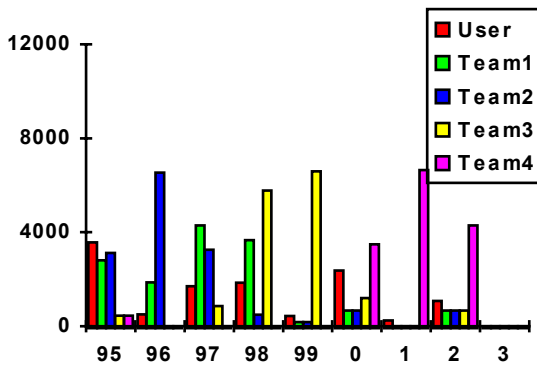


Figure 6.6-1(B): Model 6 - Expenditure of all 5 resource categories ([4,4,4,4,4] levels)

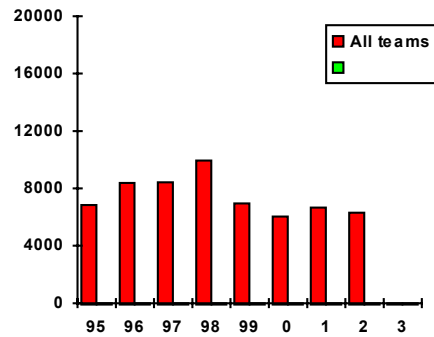


Figure 6.6-2(B): Model 6 - Overall dvp resource expenditure ([4,4,4,4,4] levels)

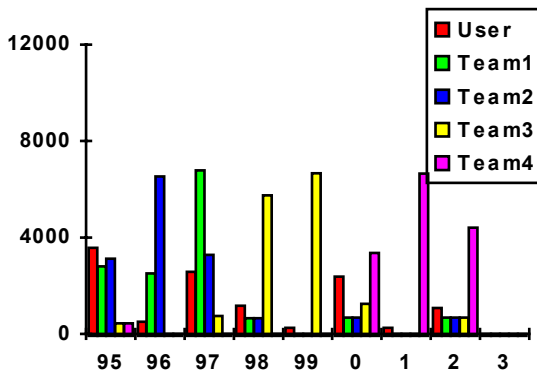


Figure 6.6-1(C): Model 6 - Expenditure of all 5 resource categories ([4,6,4,4,4] levels, constrained)

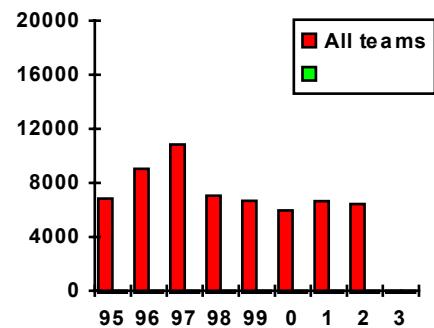


Figure 6.6-2(C): Model 6 - Overall dvp resource expenditure ([4,6,4,4,4] levels, constrained)

ID	Name	Dur.	Preds	U	1	2	3	4
1	Define overall SW structure	60d		1	1	1	1	1
2	BUILD ELEMENT 1	565d						
3	Milestones	565d						
4	SW kickoff	0d	1	0	0	0	0	0
5	URD approved	0d	...	0	0	0	0	0
6	SRD approved	0d	...	0	0	0	0	0
7	ADD approved	0d	...	0	0	0	0	0
8	Code/DDD/SUM approved	0d	...	0	0	0	0	0
9	Provisional acceptance - transfer done	0d	...	0	0	0	0	0
10	External major reviews	471.5d						
38	Software verification and validation	549d						
111	Define user requirements	80d						
115	Define SW requirements	78d						
128	Define SW architecture	78d						
141	Software user manual	39d						
144	Produce software units	69.5d						
149	Installation	12d						
152	Software transfer document	29d						
155	APPLY 1 at system level	60d	9	1	1	0	0	0
156	Build Element 2	565d						
157	Milestones	565d						
158	SW kickoff	0d	1	0	0	0	0	0
.....								
163	Provisional acceptance - transfer done	0d	...	0	0	0	0	0
164	External major reviews	471.5d						
.....								
316	Software transfer document	29d						
319	COMBINE 1&2 at system level	30d	155,163	1	1	1	0	0
320	APPLY 1-2 at system level	60d	319	1	1	1	0	0
321	BUILD ELEMENT 3	565d						
322	Milestones	565d						
323	SW kickoff	0d	163	0	0	0	0	0
.....								
328	Provisional acceptance - transfer done	0d	...	0	0	0	0	0
329	External major reviews	471.5d						
.....								
491	Software transfer document	29d						
494	COMBINE 1-2 & 3	30d	320,328	1	1	1	1	0
495	APPLY 1-2-3 at system level	60d	494	1	1	1	1	0
496	BUILD ELEMENT 2	565d						
497	Milestones	565d						
498	SW kickoff	0d	328	0	0	0	0	0
.....								
503	Provisional acceptance - transfer done	0d	...	0	0	0	0	0
504	External major reviews	471.5d						
.....								
666	Software transfer document	29d						
669	COMBINE 1-2-3-4	30d	495,503	1	1	1	1	1
670	APPLY 1-2-3-4	60d	669	1	1	1	1	1

Figure 6.6-3: Model 6 - Task sheet (durations before levelling)

7. Discussion of Results

7.1 An overview

The stated purpose of model 1 was to show how the activities identified in the ESA Software Engineering Standards are represented, how their hierarchical structure is reflected, and how sequential relationships between them are specified. Thus, one may compare Figure 6.1-1 with the outline of the standards presented in section 2.1, and with section 2.2. Also, the review processes identified in section 3.1 may be seen to be included in the figure.

The model building tool allows suppression of processes not relevant to the purpose at hand. This facility is used in the subsequent models to omit such 'supporting' processes as Project Management or Quality Assurance. Obviously it should be borne in mind that costs for such processes must be adequately provided for in any project.

7.2 Effect of product size

Comparison of models 2 and 3 shows the expected increase, by a factor of 4 roughly, in total required effort. However, the increase in duration is much less - a factor of 1.75, approximately.

The effects of levelling on project durations are interesting. For model 2, the effect is to double duration (100% increase) when levelling is on a maximum of 2 units, and to

increase by just 15% when levelling is on a maximum of 4 units. For model 3, the effect is to increase by about 34% when levelling is on a maximum of 8 units.

To investigate this phenomenon in more detail a further series of 'levelling' runs were performed for each of models 2 and 3. The results are summarised in Figure 7.2-1.

Years to completion

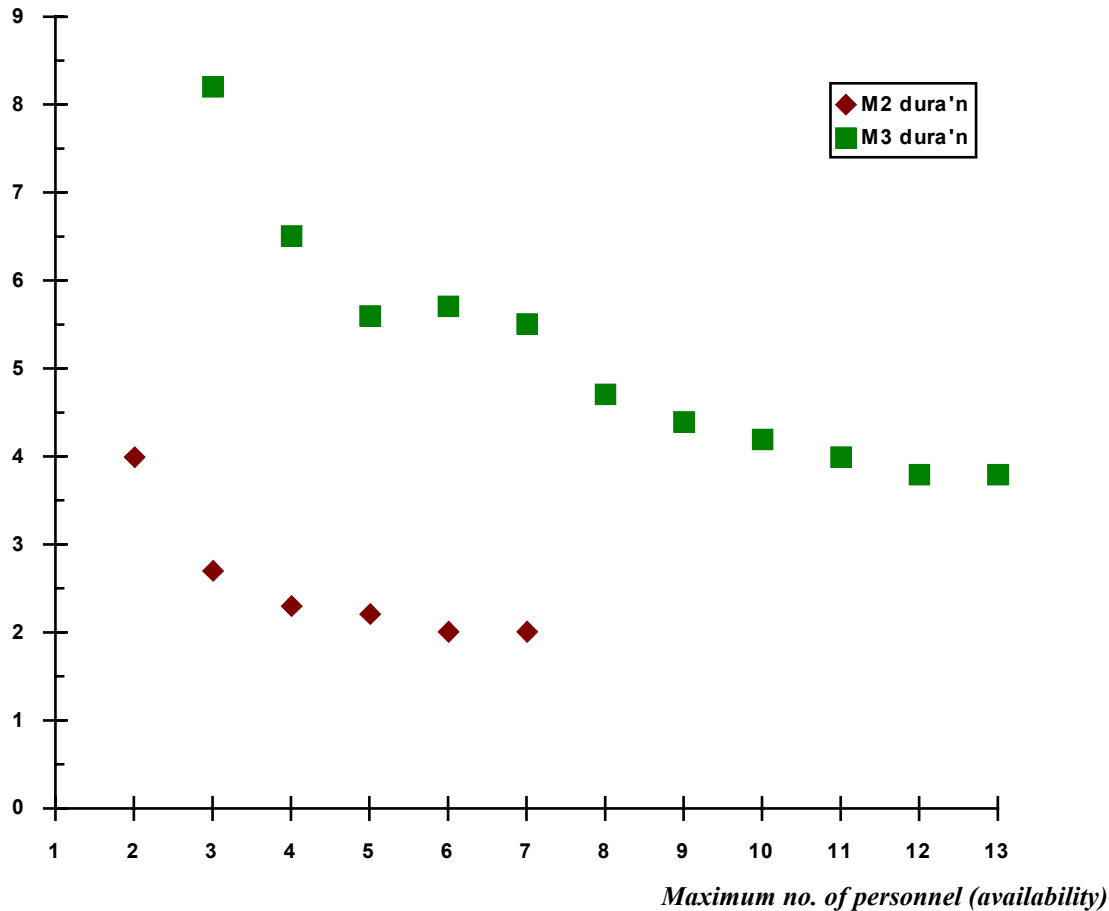


Figure 7.2-1: Effect on duration of change in maximum availability (models 2&3)

[Abscissae 2, 3 (model 2) and 3,4,5 (model 3) include some violations of the maximum availability constraint - presumed to account for any apparent anomalies in the corresponding ordinates]

From these results it seems that there is a lower limit of 'maximum availability' beyond which duration increases rapidly. This limit is about 3 units (of resource Devp) for model 2 and about 5 units for model 3. On the other hand there also appears to be an upper limit (below the ultimate 'unlimited' case) of 'maximum availability' beyond beyond which the impact on duration is relatively insignificant. This upper limit is about 4 units (of resource Devp) for model 2 and about 9 units for model 3.

It is of interest to note a parallel with findings due to Putnam, as described in Ratcliff¹⁹ (page 282). The central element of Putnam's work is the equation

$$E = LSC^3 / (C^3 * T^4)$$

where E = effort (in person years), T = development time (in years), LSC = lines of source code, and C is a constant (reflecting development environment attributes). The main characteristic of this equation is that E is highly sensitive to T, so that compressing development time will require far greater increases in effort to compensate. In fact, in practice, there is a cut-off point beyond which development time cannot be reduced.

A qualitative similarity can be seen between these findings and those reflected in Figure 7.2-1. However, the results are not directly comparable in that (a) 'Maximum availability' rather than E is used for Figure 7.2-1, and (b) it appears that Putnam's work starts from the detailed design phase. A least squares fit of the data of Figure 7.2-1 yields

$$\text{Maximum availability} = 19/T^{1.7} \text{ (for M2)}$$

and

$$\text{Maximum availability} = 153/T^{1.9} \text{ (for M3)}$$

Thus, 'Maximum availability' is sensitive to T but not to the same extent as in Putnam's findings.

In summary, one may draw up tentative practical guidelines on the basis of Fig. 7.2-1. These are presented in Table 7.2-1 but it is stressed that *they are dependent on the specific character of the underlying models used.*

PROJECT SIZE	MEDIUM [11573 hrs = 5.9 years (approx) ⁶]	LARGE [45550 hrs = 23.4 years approx.]
Recommended max. of develop. team size	4	8
Corresp. project duration	2.3 years	4.7 years
Corresp. effort expenditure profile	Figure 6.2-3, 6.2-3(a)	Figure 6.3-2

Table 7.2-1: Tentative guidelines on team size etc. w.r.t. project size

7.3 Effect of project category

Comparison of models 3 and 4 shows that suppression of intermediate internal reviews has resulted in a decrease in total effort of 6420 hours (14% of model 3 total). There is a reduction of 0.1 years (3%) in duration without levelling, and of 0.4 years (9%) with levelling (using a maximum availability of 8 units for resource Devp).

In fact, for practical purposes, it is perhaps more useful to identify the additional effort and duration incurred by inclusion of intermediate internal reviews. From this point of view, there is required an additional effort of 16% of model 4 total. The additional

⁶ Assumes 1 year = 7.5(hours) X 5(days) X 52(weeks) = 1950

durations are essentially unchanged, remaining at 3% and 9% with and without levelling, respectively.

7.4 Impact of errors detected during test

The main comparison for this aspect is between models 4 and 5. For both, intermediate internal reviews have been suppressed. Model 4 represents the limiting (and presumably unrealistic) case in which no errors are detected during any of software unit, integration or system test. Model 5 illustrates the impact of including detection of a certain number of errors at each of these stages.

Between models 4 and 5, there is an increase of 1758 hours (4%) in total effort, an addition to duration (without levelling) of 0.4 years (12% approx.) and an addition to duration (with levelling) of 0.5 years (also 12% approx.). Thus, in comparison with the previous section, the impact of 'errors detected at test' is greater than 'intermediate internal reviews' in terms of schedule but less in terms of effort. However, it should be noted that this is for just one illustration of inclusion of 'errors detected at test'; more extensive results are presented later in this section.

For this model the tasks 'System test repeats' and 'Integ[ration] test repeats' (Figure 6.5-3) both lie on the critical path. In fact, their combined durations (74.4 days and 43.9 days, respectively) account for the difference in duration between models 5 and 4 (1043.3 - 925 = 118.3 days). On the other hand, in this example, the constituent tasks of the activity 'Handle unit_test_time errors' do not lie on the critical path and so do not extend the overall project duration. These observations enable the nature of the dependence of project duration and overall effort expenditure on the number of errors detected during software integration and system testing to be investigated conveniently.

In model 4 there are 16 software units so that inclusion of error detection at unit test, with $E_U = 4$, adds a total effort of 864 hours (using the model of section A.2). Therefore, the total effort becomes $39130 + 864 = \underline{39994 \text{ hours}}$. The duration, 925 days, is unchanged.

These values of total effort and duration represent the base case of zero errors detected at software integration and system test ($E_I = E_S = 0$). Then, using the models of sections A.3 and A.4, the entries of Table 7.4-1 are calculated. These values, as dictated by the underlying models, show a linear growth of duration (without leveling) and effort with increase of each of E_I and E_S . The most striking feature is that, while the impact on effort is relatively small, that on duration may be quite significant.

E_S	E_I	0	20	40	60	
0		0%	5%	9%	13%	Duration
		0%	1%	2%	2%	Effort (Devp)
20		8%	13%	17%	21%	Duration
		1%	2%	3%	4%	Effort (Devp)
40		16%	20%	25%	29%	Duration
		3%	4%	4%	5%	Effort (Devp)
60		23%	28%	32%	37%	Duration
		5%	6%	7%	8%	Effort (Devp)

Table 7.4-1: Dependence of duration and effort on the number of errors detected at software integration (E_I) and system (E_S) test

With the essential reservation that any inferences can only be as valid as the assumptions (models) which underlie them, the following observations appear valid:

If there are substantial numbers of errors detected at software integration and/or software system test stages then the impact on overall project duration will be significant. However, the increase in effort may not be as great, possibly less than would be required by inclusion of intermediate internal reviews (section 7.3).

Thus, there may be a trade-off between effort and schedule in that one could have either

(a) an increased effort on intermediate internal reviews to result (presumably) in the existence of fewer errors for detection at software integration or software system test

or

(b) fewer intermediate internal reviews, at reduced effort, but with an extended schedule due to the existence of a larger number of errors for detection at software integration or software system test.

It is important to point out that there may well be higher level test stages, for (computer) hardware/software integration or for different levels of integration of the overall system (computer hardware & software + input/output units (e.g. sensors & actuators) + other subsystems + operators + ...). It seems clear that the impact of software errors detected at these stages will be increasingly severe. (Models analogous to those of sections A.2 to A.4 could be devised to reflect this behaviour). It is relevant, in this context, to recall the widespread belief (noted earlier) that the majority of errors are made when specifying requirements. On the other hand, the initial expenditure during software development on intermediate internal reviews (or other error-prevention and detection mechanisms) remains unchanged. Therefore, while efforts should be made to reduce this initial expenditure by streamlining the internal review processes, any proposals for dispensing completely with such review processes to meet immediate schedule pressures should be resisted as being, at best, of short term benefit only.

An underlying assumption leading to the trade-off noted in (a) and (b) above is that there is more scope for parallel work during requirements definition, architectural design and production (up to and including unit test) than there is during software integration and, especially, system test stages. This seems plausible and has been reflected in model project 5.

Finally, it is of interest to note work reported by Selby and Basili¹¹ in which they found that inspections, particularly at higher levels of design, are cost effective vehicles for error detection in comparison with 'trouble reports' (during system test or

operations). The authors observe that this is because it is more expensive to isolate errors than to fix them.

7.5 Choice of life-cycle model

The effects of resource levelling, over all resources, are quite striking (compare Figure 6.6-2(A) and 6.6-2(B)). The effort expenditure profile of Figure 6.6-2(B) is relatively even (maximum departure of 33% from annual average, standard deviation of 18% of annual average). Such a profile would seem desirable, usually, and has been achieved at a limited cost of a 16% extension of overall project duration. These fluctuations seem in line with results reported by Aoyama¹² (whose general approach appears to have some points of similarity with that of the present paper).

The imposition of an intermediate fixed date constraint results in a rather less even profile (compare Figures 6.6-2(B) and (C)). The effect of the constraint was also observed in the corresponding Gantt chart (not reproduced here) which showed, in fact, that there remains an incompatibility (even after raising availability of Team1 from 4 to 6 units). This is that activity 'APPLY 1 at system level' is started (forced) before its predecessor 'BUILD ELEMENT 1' has been completed.

In general, an advantage of an evolutionary over a waterfall life cycle model is that the former introduces more scope for concurrent work. Hence, it may provide a means of solving, to some extent at least, the 'bottle-neck' effect of higher level testing observed in section 7.4.

Finally, it must be remembered that the evolutionary model requires considerable investment in interface control (see Figure 3.2-1) and in change management. However, additional cost will also be incurred, and will be difficult to control and predict, if the waterfall model is adopted for a project for which it is not appropriate.

8. Conclusions

A characterisation of software development projects in terms of a small number of parameters and factors has been introduced. Based on these characteristics, a representative set of 6 model projects was defined.

An analysis approach based on reflecting the requirements of the ESA Software Engineering Standards has been devised, supported by a (prototype) software tool.

This approach was applied to each of the 6 model projects to investigate how variation of certain characteristics impacts on effort expenditure profiles and schedules. The characteristics included, in particular, project size, project criticality category, number of errors detected during test, and life cycle model.

On the basis of this analysis a number of tentative results, with implications for practice, were stated.

For the future, it would be desirable to apply the approach to a greater number and variety of model projects, and to compare model predictions with actual project data.

The objective would be to confirm (or not), and to make more precise, the results reported in this paper. Also, it would be of interest to apply the same approach to software engineering standards other than ESA's, as well as to the wider field of systems engineering.

Acknowledgement

The original motivation for this work came, to a considerable extent, from the author's participation in CAPTEC Ltd's development of on-board flight control software for both ESA's ISO and SOHO missions.

9. References

1. - 1991, ESA Software Engineering Standards (PSS-05-0, Issue 2), ESA Publications Division, Noordwijk, The Netherlands
2. A.-M. Hieronimus-Leuba et al. 1993, Return from Space - ESA's Technology Transfer Programme, ESA Bulletin, No. 74, pp 46-52.
3. Martin J-P. 1992, Qualité du logiciel et système qualité, Masson, Paris.
4. - 1989, Microsoft Project for Windows (Version 1), Microsoft Corporation, US.
5. Dutton J.E. 1993, Commonsense Approach to Process Modeling, IEEE Software, July Issue, pp56-64.
6. Aslaken E. & Belcher R. 1992, Systems Engineering, Prentice-Hall.
7. Potter B., Sinclair J. & Till D. 1991, An Introduction to Formal Specification and Z, Prentice Hall International.
8. Freeman M. & Beale P. 1992, Measuring Project Success, Project Management Journal, Vol. XXIII, No. 1, pp8-17.
9. Bergeron F. & St-Arnaud J.-Y. 1992, Estimation of Information Systems Development Effort: A Pilot Study, Information and Management, Vol. 22, No. 4, pp239-254.
10. Boehm, B.W. 1991, Software Risk Management: Principles and Practices, IEEE Software, January Issue, pp32-41.
11. Selby R.W. & Basili V.R. 1991, Analysing Error-Prone System Structure, IEEE Transactions on Software Engineering, Vol. 17, No. 2, pp141-152.
12. Aoyama, M. 1993, Concurrent-Development Process Model, IEEE Software, July Issue, pp46-55.
13. Kozar K.A. & Zogars I. 1992, Human and Machine Roles in Team Product Reviews, Information and Management, Vol. 23, No. 3, pp149-157.
14. Yourdon, E. 1995, When good enough software is best, IEEE Software, May Issue, pp79-81.
15. - 1986, Software Verification and Validation Plans, ANSI/IEEE Std 1012.
16. Hamilton, A. 1994, Project Management - Time drives projects, The Engineers Journal (journal of the Institution of Engineers of Ireland), October Issue, pp37,38.
17. Grady,R.B. 1994, Successfully applying software metrics, IEEE Computer, September Issue, pp18-25.
18. Fairley,R. 1994, Risk Management for software projects, IEEE Software, May Issue, pp57-67.
19. Ratcliff,B. 1987, Software Engineering: Principles and Methods, Blackwell Scientific Publications.
20. - 1992, Software considerations in airborne systems and equipment certification (RTCA/DO-178B), RTCA Inc, 1140 Connecticut Avenue, N.W. Suite 1020, Washington, D.C. 20036.

21. - 1997, Requirements for safety related software in defence equipment (00-55, parts 1 and 2, Issue 2), UK Ministry of Defence
22. - 1997, NATO quality assurance requirements for software development (AQAP-150 - *see AQAP-159 also*).
23. - 1998, Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements (IEC 61508-3, First edition), International electrotechnical Commission
24. - 1998, Guidance for FDA reviewers and industry - Guidance for the content of pre-market submissions for software contained in medical devices, U.S. Department of Health and Human Services - Food and Drug Administration - Centre for Devices and Radiological Health - Office of Device Evaluation.

Appendix: Some Modelling Details

A.1 Production of a software unit

In the model projects, this is identified as the activity 'Produce the unit' to which is usually allocated a duration of 69.5 days and an application rate of 1.25 units of software developer labour resource Devp (or Team_i (i=1,4) in case of Model 6). No allocation of resource User is made. These allocations represent a consolidation of the set of more detailed tasks listed in Table A.1-1.

No.	Task name	Duration (days)	Devp (rate)	Pred. tasks
1	Decompose a higher level design	15	1	
2	Execute internal review (general)(e.g. peer review)	5.7	1.1	1
3	Code according to a detailed design	5	1	2
4	Compile check	2	1	3
5	Document coding activity	2	1	4
6	Execute internal review (general)(e.g. code inspection or walkthrough)	5.7	1.1	5
7	Formulate unit test design	6	1	2
8	Document unit test design	2	1	7
9	Execute internal review (general)(e.g. peer review)	5.7	1.1	8
10	Define unit test spec	6	1	9
11	Document unit test spec	2	1	10
12	Execute internal review (general)(e.g. peer review)	5.7	1.1	11
13	Define unit test procedure (and harness)	6	1	12
14	Document unit test procedure	2	1	13
15	Execute internal review (general)(e.g. peer review)	5.7	1.1	14
16	Execute unit test	1	1	6, 15
17	Prepare unit test report	1	1	16
18	Execute internal review (general)(e.g. peer review)	5.7	1.1	17

Table A.1-1: Detail of activity 'Produce the unit'

From the table,

Total duration (based on the longest path), in days = 69.5

Total effort (sum of products of duration X rate, in labour days) = 87.2

Therefore, average rate of expenditure of resource Devp = 87.2/69.5 = 1.25.

For projects of Class B category (models 4 and 5), the task 'Execute internal review (general)' is excluded for all but one unit. Hence, for the activity 'Produce the unit' (reduced) one has

Total duration (based on the longest path), in days = 41.0

Total effort (sum of products of duration X rate, in labour days) = 50.0

Therefore, average rate of expenditure of resource Devp = 50/41 = 1.22.

Details of how 'Execute internal review (general)' is represented are given in section A.5.

In general, it is impossible to give an absolute definition of a software 'unit'. However, it should be clear from the schedule and resource allocations that a substantial piece of software is envisaged here. In practice, this might be further decomposed to, say, the procedure level but such substructure is suppressed for present purposes⁷.

A.2 Impact of error detected at unit test

In the model project concerned (model 5), this is identified as the activity 'Unit test repeats' to which is allocated, for each unit, a duration of 6.9 days and an application rate of 1.04 units of software developer labour resource Devp. No allocation of resource User is made. These allocations represent a consolidation of the (consecutive) set of more detailed tasks listed in Table A.2-1.

No.	Task name	Duration (days)	Devp (rate)	Per error or For all errors
1	Search for cause of the error	0.5	1	Per error
2	Correct the error (code & documentation)	0.5	1	Per error
3	Execute the test	0.5	1	For all errors
4	Prepare unit test report (i.e. analyse & report results)	0.5	1	For all errors
5	Circulate for internal review	0.1	1	For all errors
6	Make internal review comments	1.0	1	For all errors
7	Hold meeting (internal review)	0.3	2	For all errors
8	Update after meeting	0.5	1	For all errors

Table A.2-1: Detail of activity 'Unit test repeats'

From the table (E_U = average number of error detected per unit test),

$$\begin{aligned} \text{Total duration, in days} &= E_U + 2.9, \text{ if } E_U > 0 \\ &= 0 \text{ otherwise} \\ \text{Total effort (in labour days)} &= E_U + 3.2, \text{ if } E_U > 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

For model 5, E_U was taken to be 4 for all units so that total duration = 6.9 days and total effort = 7.2 labour days. Hence, the assumed rate of expenditure of resource Devp = $7.2/6.9 = 1.04$.

Note: This model is illustrative and would need to be adjusted to match practical project experience. It is important for overall project efficiency that this process be as streamlined as possible.

A.3 Impact of error detected at integration test

In the model project concerned (model 5), this is identified as the activity 'Integ test repeats' to which is allocated, as a whole, a duration of 43.9 days and an application rate of 1.02 units of software developer labour resource Devp. No allocation of

⁷ An indicative estimate for a telecommunications protocol implementation deduced from Fairley¹⁸ is for 10000 lines of (high-level) code to be produced by 60 labour months of effort. Inferring that of the order of 30% of effort is devoted to other than unit production, there remains about 42 labour months for production. This corresponds to an approximate average of $10000/(42*21.7) = 11$ lines of code per production day.

resource User is made. These allocations represent a consolidation of the (consecutive) set of more detailed tasks listed in Table A.3-1.

No.	Task name	Duration (days)	Devp (rate)	Per error or For all errors
1	Search for cause of the error	0.5	1	Per error
2	Correct the error (code & documentation)	1.0	1	Per error
3	Execute the test	1.0	1	For all errors
4	Prepare test report (i.e. analyse & report results)	0.5	1	For all errors
5	Circulate for internal review	0.1	1	For all errors
6	Make internal review comments	1.0	1.4	For all errors
7	Hold meeting (internal review)	0.3	3	For all errors
8	Update after meeting	1.0	1	For all errors
9	If necessary, execute other tests	0.5	1	Per error

Table A.3-1: Detail of activity 'Integ test repeats'

From the table (E_I = total number of errors detected by integration test),

$$\begin{aligned} \text{Total duration, in days} &= 2E_I + 3.9, \text{ if } E_I > 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

$$\begin{aligned} \text{Total effort (in labour days)} &= 2E_I + 4.9, \text{ if } E_I > 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

For model 5 E_I was taken to be 20 so that total duration = 43.9 days and total effort = 44.9 labour days. Hence, the assumed rate of expenditure of resource Devp = $44.9/43.9 = 1.02$.

The note at the end of section A.2 applies for this model also.

A.4 Impact of error detected at system test

In the model project concerned (model 5), this is identified as the activity 'Sys. test repeats' to which is allocated, as a whole, a duration of 74.44days and an application rate of 1.02 units of software developer labour resource Devp. No allocation of resource User is made. These allocations represent a consolidation of the (consecutive) set of more detailed tasks listed in Table A.4-1.

No.	Task name	Duration (days)	Devp (rate)	Per error or For all errors
1	Search for cause of the error	1.0	1	Per error
2	Correct the error (code & documentation)	1.5	1	Per error
3	Execute the test	1.0	1.5	For all errors
4	Prepare test report (i.e. analyse & report results)	1.0	1	For all errors
5	Circulate for internal review	0.1	1	For all errors
6	Make internal review comments	1.0	1.4	For all errors
7	Hold meeting (internal review)	0.3	3	For all errors
8	Update after meeting	1.0	1	For all errors
9	If necessary, execute other tests	1.0	1	Per error

Table A.4-1: Detail of activity 'System test repeats'

From the table (E_S = total number of errors detected by system test)

$$\text{Total duration, in days} = 3.5E_S + 4.44, \text{ if } E_S > 0$$

$$\begin{aligned} &= 0 \text{ otherwise} \\ \text{Total effort (in labour days)} &= 3.5E_S + 5.9, \text{ if } E_S > 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

For model 5 E_S was taken to be 20 so that total duration = 74.44 days and total effort = 75.9 labour days. Hence, the assumed rate of expenditure of resource Devp = $75.9/74.44 = 1.02$.

The note at the end of section A.2 applies for this model also.

A.5 Internal review process

In the model projects, this is identified as the activity 'Execute internal review (general)' to which is usually allocated a duration of 9 days and an application rate of 1.3 units of software developer labour resource Devp (or Team_i (i=1,4) in case of Model 6). No allocation of resource User is made. These allocations represent a consolidation of the (consecutive) set of more detailed tasks listed in Table A.5-1.

No.	Task name	Duration (days)	Devp (rate)
1	Circulate for internal review	0.2	1
2	Make internal review comments	5.0	1.4
3	Hold meeting (internal review)	0.5	3
4	Update after meeting	3.3	1

Table A.5-1: Detail- 'Execute internal review (general)'

From the table,

$$\begin{aligned} \text{Total duration, in days} &= 9.0 \\ \text{Total effort, in labour days} &= 12.0 \\ \text{Therefore, average rate of expenditure of resource Devp} &= 12/9 = 1.3. \end{aligned}$$

A reduced version of 'Execute internal review (general)' is applied within the activity 'Produce the unit' (section A.1), as indicated in Table A.5-2,

No.	Task name	Duration (days)	Devp (rate)
1	Circulate for internal review	0.2	1
2	Make internal review comments	3.0	1
3	Hold meeting (internal review)	0.5	2
4	Update after meeting	2.0	1

Table A.5-2: Detail- 'Exec. int. rev. (general) (reduced)'

from which

$$\begin{aligned} \text{Total duration, in days} &= 5.7 \\ \text{Total effort, in labour days} &= 6.2 \\ \text{Therefore, average rate of expenditure of resource Devp} &= 6.2/5.7 = 1.1. \end{aligned}$$

The foregoing are typical definitions that focus on schedule and overall resource allocations. Evidently these aspects will need to be modified for different projects and organisations. Also, considerable thought must be given to the detailed procedures and

mechanisms for conducting such reviews (see e.g. Kozar & Zogars¹³) if they are to be efficient and effective.

A.6 Defining SW requirements and architecture

The activity 'Define SW requirements' (for example, activity 98 of model 2 (Figure 6.2-4)) is specified in terms of 2 parameters:

S_B = Number of branches per logical model (Chapter 3¹) component
 S_D = Number of decomposition levels in logical model (Chapter 3¹)

Thus, the logical model for $S_B = 4$, $S_D = 2$ has the tree structure depicted in Figure A.6-1:

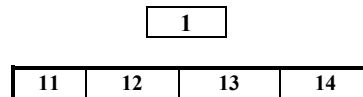


Figure A.6-1: Tree Structure for 4 Branches & 2 Levels

On the other hand the logical model for $S_B = 4$, $S_D = 3$ has the tree structure shown in Figure A.6-2:

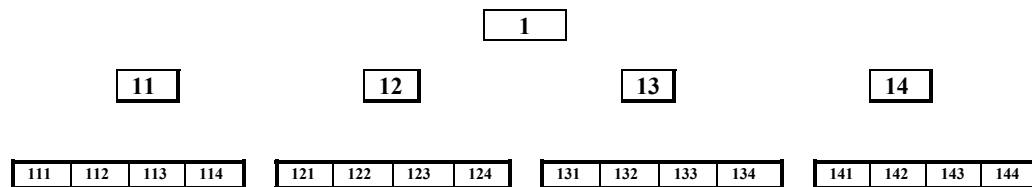


Figure A.6-2: Tree Structure for 4 Branches & 3 Levels

Clearly, these models are idealisations in that such symmetric decompositions can hardly be expected in practice. However, for present purposes, they provide an adequate characterisation of size and structure.

At each node three consecutive tasks are performed,

Construct a logical model	(Duration 20 days, Developer 1 unit)
Specify software requirements	(Duration 10 days, Developer 1 unit)
Execute internal review (general)	(Duration 9 days, Developer 1.3 units)

In exactly the same way the activity 'Define SW architecture' is defined in terms of 2 parameters:

A_B = Number of branches per physical model (Chapter 4¹) component
 A_D = Number of decomposition levels in physical model (Chapter 4¹)

The parameter N_U (= Number of software units (or modules - Chapter 5¹)) is taken to be equal to the number of lowest level nodes (16 in case of $A_B = 4$, $A_D = 3$).

At each node three consecutive tasks are performed,

Construct a physical model	(Duration 20 days, Developer 1 unit)
----------------------------	--------------------------------------

Specify architectural design	(Duration 10 days, Developer 1 unit)
Execute internal review (general)	(Duration 9 days, Developer 1.3 units)