Dublin City University
School of Computer Applications

Implementing Metrics For Process Improvement

Angela B. McAuley

A thesis submitted as a requirement for the degree of
Master of Science in Computer Applications.

August, 1993

Supervisor: Renaat Verbruggen

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master of Science in Computer Applications, is entirely my own work, and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Angela Mc Auley..... Date: 20th September, 1993.....

Acknowledgment

I would like to thank my Supervisor, Renaat Verbruggen, for his encouragement and guidance throughout the course of this research. I am grateful to Michael O'Callaghan, General Manager, Microsoft Ireland (Worldwide Product Group), who gave me encouragement and support, in terms of time and resources to complete this project, over the past two years. I would like to take this opportunity to thank all those within Microsoft Ireland who were involved either directly or indirectly with implementing the metrics system over the past year, for their valuable help and comments, particularly those on the 'metrics team' who devised the initial set of measures.

I would like to thank my parents, Michael and Rosemary, for their constant encouragement and support throughout the course of this project; my friends for their everlasting patience, and my siblings for their help with the house and garden, while I was writing up this thesis.

Implementing Metrics for Process Improvement

Angela McAuley

Abstract

There is increasing interest in the use of metrics to control the software development process, to demonstrate productivity and value, and to identify areas for process improvement. Research work completed to date is based on the implementation of metrics in a 'standard' software development environment, and follows either a top-down or bottom-up approach. With the advent of further European unity, many companies are producing localised products, ie products which are translated and adapted to suit each European country. Metrics systems need to be customised to the processes and environment of each company. This thesis describes a 12-step process for metrics implementation, using an optimum approach which is a combination of top-down and bottom-up approaches, with a set of applicable metrics, covering the software development process, which can be adapted for any development environment. For the case study, a software localisation company, the suggested implementation process is followed, and relevant measures are adapted to suit the different environment, with a particular emphasis on quality metrics. This thesis also demonstrates that a metrics system is itself subject to continuous improvement, and rather than being a once-off implementation, it is an evolutionary process, changing as the software development process comes under control.

CONTENTS

1. PREFACE

> *"When you can measure what you are speaking about and express it in numbers, you know something about it;*
> *but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind."*
>
> [Lord Kelvin]

When speaking of metrics with respect to software, computer professionals often think in terms of code-level metrics. Most of these metrics surfaced in the late 70's and early 80's, the most widely used ones are the factor analysis models such as COCOMO [Boehm81] and complexity models [Halstead77]; [McCabe76].

During the 1980's there was much research conducted into the usefulness of these metrics. Now the 1990's has become a decade where the Quality of software is the most important factor concerning today's developers and users. The subject of metrics has expanded considerably since the effort and complexity metrics first emerged in the 1970's.

Quality is a concept which has been with us in the Western world for the past 2 decades, but mainly in the manufacturing area. Up until recent years, Software development was viewed as a 'black art', and therefore any attempt to measure the quality of the process or the resulting product was deemed to interfere with the creativity of the developers. The primary concern of software companies is to develop a process to systematically and consistently produce Quality software.

In order to improve the process, it needs to be measured, otherwise, as the process changes, there will be no objective means of determining whether or it is improving. *What gets measured gets done... Even imperfect measures provide an accurate strategic indication of progress, or lack thereof.* [Peeters89]. The introduction of a Metrics system is an evolutionary process, starting with basic global measures, and moving towards more detailed measures as time goes by.

De Marco said "you can't control what you can't measure". [DeMarco82] I'd expand on this to state "you can't improve what you can't control". Therefore, companies must firstly measure to control their processes, then use these measures to improve.

2. INTRODUCTION

Implementing a comprehensive set of useful measures is a difficult task. The measurement system should provide data to help with project planning, process and project control, and intra & inter project analysis. One reason many measurement programs fail is that they are implemented in a 'quick-fix' manner, without focussing on the main issues. The measurement system should fit into an overall framework, with the following aims:

- Enable effective Planning
- Enable control of the process
- Determine project progress
- Enable benefits of new tools/methods to be stated in quantifiable terms
- Identify areas in need of improvement
- Enable improvement goals to be set in quantifiable terms
- Demonstrate improvement

Metrics are normally introduced in a company within the context of process improvement - a quality culture provides a positive environment for measurement. The concept of process maturity is introduced in this chapter, which provides the background information and understanding for the process developed, to implement metrics that satisfy the aims listed above.

Metrics are a static picture of the process at a point in time, and will provide an indication of the current health of a process or product. Measures themselves will not improve the process. My dissertation describes a method for the successful introduction of an initial set of measures, which will provide quantitative information on the important process areas, but a process improvement program needs to be addressed as a separate entity, to be initiated in conjunction with the metrics system. Thus both the qualitative (organisational culture, improvement teams) and quantitative (metrics) sides of Quality Management would be addressed.

Metrics should cover the entire product development cycle. They are firstly used for initial estimating of a project, in terms of effort, time and cost. As the project progresses, measures are used to keep track of effort, time and cost, as well as measuring the quality of the product. Finally, measures are used at the end of the

project, to measure total time, effort and cost against the original estimates, and to quantify the quality of the developed product.

Traditionally, the time and cost for a project is first estimated, then development of the software commences. Invariably, the original estimates are way off. One way of getting closer estimates is to first estimate what it should ideally take, then multiply this estimate by a factor, say 2. One still won't be able to substantiate these estimates to customers, though. So why can't software companies seem to get it right? There are several reasons put forward by [CSE92]: lack of expertise; new projects; personal bias; lack of standards; lack of data; constraints, and political decisions.

Lack of expertise is directly related to lack of practice. Because egos are often involved, people tend to underestimate when it comes to estimating the time or cost of the project. Estimates fall prey to the 'wishful thinking' scenario.

New projects refers to the fact that there is no data from previous experience to go by, hence there is nothing there to measure by. The estimator must start from scratch without knowing the potential project and planning risks.

Personal bias ties into the lack of expertise area described above - people often think that they are better than they are, or subconsciously want to complete something in a shorter time than someone else, hence the estimation is incorrect.

Lack of standards refers to ambiguities and subjective decisions that are a result of depending on the individual project managers, rather than on a stable process.

Lack of data refers to the lack of historical data, either because the project is new, or because there are no measures kept historically. This ties in with the 'new projects' category above.

Constraints refer to the numerous constraints upon a project, particularly with respect to a release deadline, or resource constraints.

Political decisions in this context refer to an instance where a manager will decide to make a loss-making bid, and then force the estimates to match the bid figures.

2.1 Software Management

Projects must be on time and within budget, and not contain bugs. Planning the project, then measuring what actually happened, against the plan, is the key to success. This dissertation shows how an initial metrics system can be implemented in order to gauge where a company is now and then to set targets for where they want to go, as part of a corporate process-improvement program, which should be initiated around the same time as the metrics system. Success comes from detailed prior planning and using measurement to learn from each project. Measurement in itself is not a silver bullet, but is an essential factor in process improvement activities, which will also include qualitative factors such as introduction of Total Quality Management concepts, and Process Improvement teams.

The Management puzzle contains several interdependent pieces, which can be categorised under the headings, of People Management, Resource Management, Process Management, and Quality Management [Choppin91]. These categories have been amended, below, to suit the software development environment:

The People Management category could contain sub-categories such as personal motivation, teamwork, communication, customer attitudes. Measures to use here are the subjective, 'soft' measures, which measure perception and attitude.

Resource Management covers management of all resources - money, machines, manpower, etc. This category includes sub-categories like project planning, productivity, configuration management, hardware and software environment, data control, cost and schedule control.

Process Management includes areas such as tools, techniques, methodologies, standards, test management, acceptance testing.

Quality Management refers to the management of the quality system, and ties in with the other three categories. This area should include sub-categories of cost and schedule estimation, quality planning, bug analysis.

> *The central problem of management in all its aspects ... is to understand better the meaning of variation, and to extract the information contained in variation*
>
> [W.E. Deming]

2.2 The Capability Maturity Model

The Capability Maturity Model (CMM) [SEI91-TR24] was developed by the Software Engineering Institute, based in the Carnegie Mellon University. The framework was originally developed to enable the United States Department of Defense to assess the capability of their software contractors. The first draft was available in September 1987, and was amended and updated several times, to produce version 1.0 which was published in August 1991. Both the Software Capability Evaluation program (used by US government agencies to assess contractors), and the process assessment program (used by organisations to assess their own process) were developed in parallel. The Capability Maturity Model is designed to provide organisations with guidance on how to gain control of their process for developing and maintaining software and how to evolve towards a culture of software excellence. It does this by serving as a model against which an organisation can determine its current process maturity and by identifying the key issues critical to software quality and process improvement.

The CMM contains five levels of Software process maturity: Initial, Repeatable, Defined, Managed and Optimising. Each level contains a list of key practice areas, which include many of the management areas listed in section 2.1 above. Full details of the CMM can be found in [SEI91-TR24], an overview of each level is provided here.

Level 1, Initial:

An organisation at this level is without a stable environment for developing and maintaining software. Controlled progress in process improvement is not possible, as there is no metrics system in place. There are few stable software practices in place, and performance is predicted by individual, rather than organisational, capability. In other words, a project might be successful, but if the Project Manager leaves, or some key individuals in the team move to another team, the earlier successes are unlikely to be repeated.

Level 2, Repeatable:

An organisation at this level has some basic measures in place i.e. stable processes are in place for planning and tracking the software project. Costs, schedules and functionality are tracked, and problems in meeting commitments are identified

when they arise. Software configuration management procedures are used. Project standards exist, and the software QA group ensures that they are followed. There is a stable, managed working environment. The Key Process Areas for Maturity Level 2 are: Requirements management, Software project planning, Software project tracking and oversight, Software sub-contract management, Software quality assurance, and Software configuration management.

Level 3, Defined:

An organisation at this level has a standard process for developing and maintaining software across the organisation. The process no longer depends on individuals for success. A Software Engineering Process Group facilitates software process definition and improvement efforts. An organisation-wide training program is implemented to ensure that both staff and managers have the knowledge and skills required to see their responsibilities through. Projects use the organisation-wide standard software process as a template for their own project. Each project uses a peer review process to enhance product quality. The Key Process Areas for Maturity Level 3 are: Organisation process focus, Organisation process definition, Training program, Integrated software management, Software product engineering, Intergroup co-ordination, and Peer reviews.

Level 4, Managed:

An organisation at this level sets quantitative quality goals for software products. Productivity and Quality are measured for important software process activities across all projects in the organisation. There is a comprehensive metrics system in place, measuring both the software products and process, which is seen as part of the process, rather than an additional activity. The data is gathered automatically wherever possible, as manually collected data is subject to error. The metrics are analysed and the results used to modify the process to prevent problems and improve efficiency. The Key Process Areas for Maturity Level 4 are: Process measurement and analysis, and Quality management.

Level 5, Optimised:

An organisation at this level focuses on continuous process improvement. The organisation identifies the weakest process elements and strengthens them, with the goal of preventing the occurrence of defects. Metrics concentrate on the process rather than on the product. Continuous process improvement is a result of working on the areas identified from the process metrics information, and testing innovative ideas and new technologies. At this level, defect prevention is the all-encompassing goal of the organisation. The Key Process Areas for Maturity Level 5 are: Defect prevention, Technology innovation, and Process change management.

The Capability Maturity Model is useful for determining the areas of a company's process that require most improvement. In theory, a company should fit neatly into one of the five maturity levels. In practice, different parts of the software process could be at different levels, or two projects could be at different levels. Terry Bollinger and Clement McGowan [Bollinger91] conclude that the assessment rating system is seriously flawed due to its reliance on the unproven maturity model. Their two main points are that the model itself favours maintenance processes with relatively narrow product definitions, and that the information recorded as a result of assessments is limited - the final reports do not show how the process is structured and controlled. Their final recommendation is that whilst the process assessment program itself has made an outstanding contribution to the software industry, the current grading system of five distinct process-maturity levels is so fundamentally flawed, that it should be abandoned.

In response to the Bollinger-McGowan article, Watts Humphrey and Bill Curtis [Humphrey91a] defend the process maturity model and the assessment questionnaire. They state that the process maturity model is based on the widely-accepted quality improvement principles of W.A Shewhart, W.E. Deming, and J.M. Juran, and therefore it is a proven model. They say this framework, like the other models widely accepted in manufacturing industry, models the stages that an organisation must go through to establish a culture of engineering excellence. Each model stage lays the foundation on which effective practices for the next level are built.

My own views on the Capability Maturity Model are that it has both good points and points which need to be addressed further. The fact that the maturity model is itself subject to continuous improvement is a good one, and I believe the current issues people have with it will be ironed out in due course.

Points for:

- It's a good place to start - by determining a company's maturity level, some obvious immediate improvement opportunities are recognised.
- Similar to evolutionary models available for other industries, which have been proven.
- Key Practices information suggests interim improvement goals and progress measures, which makes it easy to use as the basis of a company's improvement program.

Points against:

- Developed for US DoD, with large systems, and needs some adaptation for Irish/European software industry.
- There is too large a gap between some of the levels - eg between 3 and 4. The levels could be further defined, so that they each level can be attainable within say two years of reaching the previous level.
- The Key Practices are too distinct, by belonging to just one maturity level. eg Metrics are only mentioned at level 4, whereas different levels of metrics should be used from level 1 upwards.

Overall recommendation:

- Use the Capability Maturity Model as an initial guideline to gauge the company's current process level. It provides some good pointers for companies at the lower maturity levels.
- Once the base level is determined, the correct level/granularity of measures to use should be chosen. The higher the capability level, the more detailed are the measures required.

Watts Humphrey uses the example of Hughes Aircraft [Humphrey91b] to describe the successful implementation of a process-improvement program with respect to the Capability Maturity Model. This program saw the Software Engineering Division of Hughes Aircraft move from level 2 to a strong level 3. The article demonstrates that the overall objective of the Capability Maturity Model is to achieve a controlled and measured process as the foundation for continuous improvement.

Summary

The Capability Maturity Model is a useful framework to provide a guideline for process-improvement activities. This dissertation concentrates primarily on metrics to introduce in a company at level 2, and also provides guidance for companies at levels 1 and 3. Metrics for companies at the higher levels of 4 and 5 are not included as it is not intended to demonstrate how a company can move from one level to the next, through the use of metrics. To enable a company to identify the tasks currently requiring most improvement, and to quantify any resultant improvements, suitable measures are required throughout the software development process. Section 3.4.2 lists the types of metrics that could be used at each of the maturity levels 1 to 3.

2.3 Defining Measures To Facilitate Process Improvement

Measurement is a critical factor in quantifying the current health of the development process. The process involved in implementing a successful metrics system, to facilitate process improvement is an evolutionary one, as it is a cultural change that needs to be effected. It would be naïve to expect it all to work out at the first go. If a company starts simple, and builds on the metrics as time goes by, and as the employees come to fully understand their usefulness, then the system will be successful and will support any continuous improvement efforts. To support evolution, [SEI91-TR16] states there is a need to plan for regular reviews of all aspects of the measurement program (goals, implementation, use, delivery, cost effectiveness). The authors report that the most successful programs they observed supported experimentation and innovation. Follow the process described in chapter 3 to implement the initial metrics system, then start to improve the process in accordance with whichever Quality philosophy is chosen. The measures should be amended as each company sees fit, as the process comes under control and improvements are made. What is described forms the corner stones for the initial measurement of current projects and processes, assuming a process improvement program will be introduced either in parallel or subsequently.

The Pyramid Consortium state that there are three vital things to remember when introducing metrics [Pyramid91] - Be goal oriented, Use simple global indicators, and Be patient.

1. *Be goal oriented.* Always start from quality or productivity improvement objectives. Measurement should only be a means to evaluate attainment of these objectives. Measurement on its own, without related goals is no use.
2. *Use simple, global indicators.* Start small, and stick to global indicators which cover the whole process. Some companies start with just one measure - defects. Other companies will start with measures in several categories. Typical measures to start with are effort (manhours), size/complexity (lines of code, function points), and quality (defects).
3. *Be patient.* Some companies expect to see improvements instantaneously. However, it takes time to go around the feedback loop. Typically, according to Pyramid, it takes at least two years before the first big improvements in quality and productivity are properly established.

For software metrics,

> *"The goal should be a set of measures*
> *that can be justified theoretically,*
> *that can be supported empirically, and*
> *that can be used with confidence by both*
> *programmers and project managers."* [Shen83]

2.4. Summary of Metrics w.r.t the Process Maturity Framework

The Contel technology Centre [Pfleeger90] suggests a set of metrics for which data should be collected and analysed, based on the Capability Maturity Model. The article recommends that metrics are to be implemented step by step in five levels, corresponding to the maturity level of the development process. Level 1 metrics provide a baseline for comparison as improvements are sought. Level 2 metrics focus on project management; level 3 metrics focus on the products developed; level 4 metrics measure the process itself in order to control it, and level 5 metrics focus on the process with feedback loops in order to change and continually improve the process.

The Software Engineering Institute have written a Technical Report which describes in detail appropriate measurement indicators to use for each of the key process areas for each maturity level [SEI-92TR25]. They have chosen to discuss measurement indicators, (ie type of measure) rather than specific documented metrics (ie what measures to use and how to use them). The reason is that, for example, an indicator of size allows a discussion of trends in size changes and their implications, whether size is measured using lines of code, function points or pages of documentation.

The following table shows a summary of the process maturity framework, with some level-appropriate metrics, based on a chart from [Pfleeger90], which has been adapted based on my interpretation of the applicable measures.

Level	*Characteristics*	*Metrics*
5 - Optimising	Improvement fed back to process - defect prevention	Measurement of all processes (on a continuous, integrated basis) rather than products, to facilitate continuous improvement of the development process.
4 - Managed	Processes are measured, with feedback from early activities feeding into later activities	Measurement of processes begun to allow greater control of these processes. Measurement and control of sub-processes.
3 - Defined	Processes are defined and stable	Product measures, including detailed internal and external quality and usability measures
2 - Repeatable	Process dependent on individual experience/ expertise	Project Management metrics (schedule, effort, progress curves, productivity, product stability, complexity and quality measures)
1 - Initial	Ad Hoc	Baseline metrics to identify areas most in need of improvement and to demonstrate improvement in these areas as time goes by (eg product size, total effort and basic quality metrics, ie bug analysis)

In theory, segregating measures into the five maturity levels is a good idea. However, I think that the measurement types should not be so strictly categorised, and should have some product measures at level 2, some project management measures at level 3, and some process measures at levels other than at level 4.

Since this dissertation describes a process for implementing an initial set of measures, the assumption is that most readers belong to companies at the lower levels of process maturity. The maturity distribution of 59 sites, representing 296 projects [SEI92-TR24] shows 81% of the sites at level 1, 12% at level 2, and 7% at level 3. Therefore, the measures described in chapters 4 and 5 are primarily those suggested for companies at levels 1, 2 or with some projects at level 3. Any companies at level 4 would already have a complete, successful metrics system in operation.

3. METRICS IMPLEMENTATION PROCESS

There are twelve steps in the metrics implementation process. The order in which they are here is that which should give the best outcome, although a few of the steps could be interchangeable, (eg steps 10 & 11), depending on the company and its metrics goals. The 12 steps are:

1. Map the software development process.
2. Define the corporate improvement goal
3. Conduct an employee and a customer survey
4. Define applicable metrics categories
5. Break corporate goal into a specific goal for each applicable category
6. Define specific measures
7. Prepare data sheets
8. Provide necessary training
9. Measure current process and products
10. Set improvement targets
11. Automate the system
12. Review the effectiveness of the metrics

Each of these steps is described in detail, quoting appropriate references where applicable. A description of how the steps were implemented in a Software Localisation environment follows this.

3.1. Map the software development process

In order to successfully improve, and to direct all efforts in the same direction, companies must first clearly define what they are doing. Until everyone fully understands the current process, and its inherent problems, the improvement efforts will be mis-directed, with wasted effort on areas of little importance to the overall process. For initial success, it is important to concentrate on areas where the improvements will make most gain. Using the Pareto principle, one should concentrate on those areas where 20% of the effort will produce 80% of the gain.

3.1.1 Define The Process

Watts Humphrey [Humphrey88] states that an important first step in addressing software problems is to treat the entire development task as a process that can be controlled, measured and improved. He defines a process as a sequence of tasks that, when properly performed, produces the desired result.

Within the context of Total Quality Management, a process is defined as:

> *"A repetitive and systematic series of legitimate actions or operations on an input directed toward the achievement of a goal or outcome"* [Qualtec91]

The components of this definition can be further clarified:

Legitimate:	Individuals have clear responsibility for the process, which has been authorised through the appropriate channels.
Have Inputs/Outputs:	Every process must have both inputs and outputs. These can take the form of people, material, equipment or methods. The output, or goal of the process is clearly defined and measurable.
Have actions or operations:	A clearly defined set of activities is associated with the process. These activities can be observed and defined.

Be repetitive:	The actions and/or operations are ongoing and are performed on a regular basis.

At this stage, the full project life-cyle for the company should be defined This is the highest-level view of the company's process. Each company will identify its own life-cycle phases, depending on its particular business, but in general, the following major process stages will be present.

Requirements definition; Design; Development; Testing, and Implementation.

Each company has its own definitions for the activities within each of these process areas, and its own methods. In very generic terms, the phases can be described as:

Requirements Definition: During this phase, the customer requirements are analysed and quantified. The overall system concept is developed, together with a development plan and a resource plan. A project specification document is produced.

Design: Once the project has been approved, the design phase is entered. Firstly, the overall system design is developed, from which detailed program and module designs are developed.

Development: This phase primarily involves coding the system as designed in the previous stage.

Testing: The testing phase involves a number of sub-phases. These are unit test, integration test, and system test. The code is debugged during the testing process.

Implementation: This phase involves implementing the fully functional system for the customer, and includes an acceptance testing process.

When the life-cycle has been defined, a company can progress to defining the process in more specific detail.

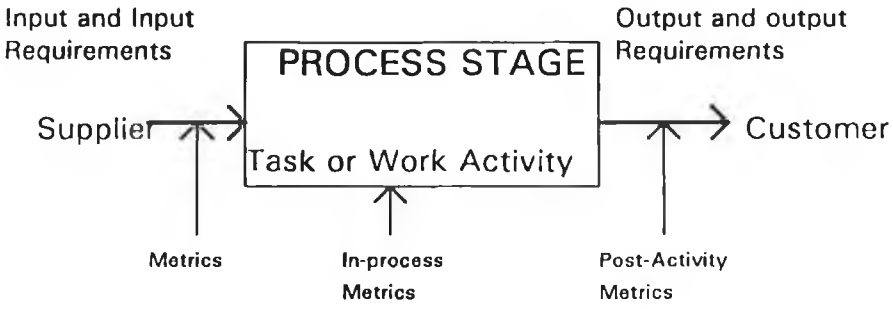
The following steps describe how to map the process:

1. Define each of the activities in the process, and who is responsible for completing each activity.
2. Define the inputs to and outputs from each activity.
3. Define any decisions that are made between activities.
4. Finally, chart the process, from start to finish.

Sandhiprakash Bhide [Bhide90], presents a process stage model that is a natural extension of the customer-supplier process model outlined above. The process exists of the following elements:

Process stage:	A state of evolution of the product, eg. designing, testing, implementation.
Task or Work Activity:	The specific activity that transforms the process inputs to the process output.
Suppliers:	The Supplier supplies the necessary inputs to the process under defined input conditions
Customers	The Customer receives the output of the process in accordance with specified output requirements
Input and Input Requirements	The inputs in the requested input format are necessary to perform the task or work activity and produce the output in the output format
Output and Output requirements	These are what the customers define as the necessary requirements

Process Stage Diagram



3.1.2 Chart The Process

It is easier to interpret the process if the flowchart is drawn/arranged either as a conventional flowchart, as a PERT network, or within a frame of rows and columns - the rows representing each process stage, and the columns representing each team involved in the process, although this can be very time-consuming to complete.

Using different shapes/sizes/colours of Post-It notes for the exercise makes it quite easy to accomplish the task of process mapping, as Post-Its can easily be stuck onto a large wall, table or board and rearranged without causing delay or rework. Once the Post-Its are in place, the map layout can be transferred to a more permanent form, either on paper or using a suitable software package.

With the process charted, the weak links in the process can be more easily identified. The graphical representation helps show process areas and support structures that are more stable than others. The Capability Maturity Model helps to further classify the areas that need to be improved, and prioritises these by arranging them in the appropriate level (ie achieve objectives from the lower levels first).

The level of detailed required in the process chart depends on at what level of detail the process is still global. At a very detailed level, there may be differences between projects in terms of tools and specific methods employed. The aim is to achieve the lowest common denominator across all projects, and map this process.

The following chart shows a portion of a high-level view of a sample software development process. A PERT network is more suitable and easier to produce, for a more detailed process chart.

Department/ Process stage	**Customer**	**Development**	**Testing**	-
Req. Defn.	Define Requirements and review specification	Develop requirements specification	Develop draft Master Test Plan	-
System design	Review Master Test Plan	Develop Design Specification	Develop detailed Master Test Plan	-
Program Design	-	Develop program structure and module specifications	Develop integration test spec and system test spec	-
Programming	-	Code and debug	Module Test	-
Integration Test		Debug	Test w.r.t. integration test spec and report bugs	-
-	-	-	-	-

3.1.4 Identify Current Maturity Level

Once the overall development process has been mapped, the appropriate maturity level of the processes needs to be identified. Therefore, the next task after mapping the process is to ascertain the company's process maturity level.

The Capability Maturity Model [SEI91-TR25] identifies key process areas for each maturity level. This initial CMM documentation states that the key process areas of a particular level must all be satisfied before the next stage of maturity can be commenced. A company currently at level 1 would need to work on the key process areas of level 2, and to be fully in control of each of these areas before moving on to tackle the key process areas of level 3. Section 2.2 explains that this is not necessarily so in practice - different parts of the software process could be at different levels, or two projects could be at different levels. This is also the assumption made in a later SEI technical report [SEI-92TR25], which shows how individual measures can be applied to each of the Key Process areas. This document also states that a metric which is discussed in the context of a higher maturity may be used by a project at a lower maturity, although the data may be less accurate. I strongly agree with this newer approach, and most of the suggested measures (used from levels 1 through to 3) are based around the key process areas for level 2, the repeatable level, ie project planning, tracking, and Quality Assurance.

The Key Process Areas for level 2, repeatable:

Software Configuration Management
Software Quality Assurance
Software Sub Contract Management
Software project Tracking and Oversight
Software Project Planning
Requirements Management

3.2. Define the corporate improvement goal

At this stage, the current process is known, with an idea of the current maturity level, and some of the areas that should be concentrated on. All process improvement books and papers categorically state that before proceeding with any form of process improvement initiative, a company must firstly define a goal for itself. This section describes the establishment of the overall corporate improvement goal, and section 3.5 defines the specific improvement goals to a greater degree of granularity. Tom Peeters [Peeters89] says that it is imperative to consider bold goals for the corporate improvement programme. He goes on to state:

> "Such non-incremental goals, which will require you to 'zero-base' the business and seek completely new ways of organising everything - from accounting systems to organisational structure to training to equipment layout and distribution network relations - are a common-place necessity today."

3.2.1 A Sample Goal

A company may have a general process-improvement goal, or a more specific corporate strategic goal. This goal then needs to be further broken down into smaller more specific goals. Metkit is one of the CEC sponsored ESPRIT projects, which has devised a 'kit' for the implementation of metrics, and concentrates on the area of metrics education using case studies. A good general goal from the Metkit Consortium [Metkit92a] that they suggest should be applied to all companies is:

Continuous improvement of all processes which leads to better use of our resources, improved efficiency of our processes, improved productivity of our project teams and improved quality of our products.

The bottom line is customer satisfaction. The corporate goal should strive towards complete customer satisfaction - this effectively means reducing errors, being more productive, and less costly.

As an example of a bold long-term goal that works, Motorola has a quality improvement program that it applies to all of its operations, called Six Sigma. Its goal is to reduce defects in all areas of operation to 3.4 defects per million units.

Dwight Davis [Davis92b] reports that Motorola have come up with new schemes to quantify defects in IS operations. The example quoted is that instead of tracking system availability on a percentage uptime basis, they track how many transactions fail to occur during narrow time slices throughout the day. The number of transactions that don't get done becomes their defect unit for that measure. The success rate quoted to date is to increase system availability from 4 sigma to 5.5 sigma since the Six Sigma IS program began in 1988, which is a move from 6,210 to 32 defects per million units, and in percentage terms is an increase in uptime from 96.3% to 99.98%.

3.2.2. The Goal - Question - Measure approach

The Goal-Question-Measure approach, originally developed by Victor Basili, University of Maryland, for evaluating defects for a set of NASA projects, is recommended by the Metkit consortium [Metkit92a]. It is a philosophy of software measurement which uses a top-down approach in order to put useful, meaningful measures in place. It ensures that the metrics that are implemented relate to the corporate goals and are not superfluous or non-relevant. Two of its main strengths are that it does not rely on any standard measures and it can cope with any environment.

a) Goal

The goals should be defined in terms of

- what is wanted
- who wants it
- why it is wanted
- when it is wanted

An example would be 'to improve the quality of our released software to maintain our client base, over the coming two years'

b) Question

The goal is then refined into a set of questions that require quantifiable answers Some questions for the above goal would be 'what are our current pre-release and post-release defect rates?' 'How much effort do we spend on bug fixing?'

c) Measure

Finally, the qualitative goal is transformed into a quantifiable goal

The above goal, 'to improve the quality of our released software to maintain our client base, over the coming two years', could be rewritten as: 'to halve both our pre-release and post-release defect rates within two years, with a corresponding reduction in bug-fixing effort'. The measures to be put in place would therefore be pre-release defect rates, post-release defect rates, and bug-fixing effort.

The AMI project (Application of Metrics in Industry) is also an ESPRIT project based on the goal-oriented approach, in order to allow the metrics to be defined according to each company's goals [AMI92]. The main benefit of the AMI approach is that it is a tried and tested method which couples the use of metrics to the achievement and improvement objectives of an organisation. This is also a top-down goal-based approach consisting of four main activities:

- Assess the environment to allow for the primary definition of goals
- Analyse the assessment conclusions to build the goals tree and to identify the most suitable metrics
- Metricate by implementing a measurement plan
- Improve as the measures are exploited and actions are implemented

Establishing the high-level corporate goal is the starting point, but before specific goals can be established, all employees and the most important customers should be involved in establishing the current state of the process.

3.3. Conduct an employee and a customer survey

The more information collected upfront, the better the direction that can be provided to the improvement activities. Involving as many people as possible in the early stages enables a solid baseline to be established from which to improve and to measure these improvements.

In order to make a journey, three things are required - a known starting point, a known objective, and a way of making the journey. A map on its own is of no use.

If you don't know where you are, a map won't help [Watts S. Humphrey]

It is necessary to pinpoint the current location on the map, before one can see the direction in which he/she should travel in order to reach the chosen destination. All companies do some things exceptionally well, some things very badly, and manage somehow to get through the rest. It is very beneficial to know which working practices come under which heading. There is no point in spending large amounts of effort in making small improvements to good practices, when others are in a mess. Neither is it a good idea to spend time and effort on improving non-important areas that will show no tangible benefit to the company.

Therefore, a health-check is required. The first thing to do is to identify who and what should be included. The health check would be in the form of surveys and would concentrate on the following areas:

1. What is our customers' view of the quality of our service?
   (This is a customer evaluation of the service to them)

2. What is the staff's understanding of the meaning of quality w.r.t. our process?
   (What impact do project planning, risk assessment, etc., have on quality now, and how important are these perceived to be by the company employees?)

Conducting a survey serves several purposes. Firstly, it establishes the baseline, ie provides 'soft' measures, and helps focus effort on measuring the areas that matter most. Secondly, it involves each employee and thirdly, it gives valuable feedback from the customers.

3.3.1 Customer View

No matter what a company's business, the most influential factor on whether or not it will be successful is its customers. A company might be a world leader in project planning, cost and productivity rates, and at the same time have customers that are extremely dissatisfied with their services. It is possible to be brilliant at doing the wrong things, and therefore if a company is to flourish, it must ensure that its customers are fully satisfied with its services.

Customer satisfaction can be very difficult to measure precisely and consistently, by its very nature of being a subjective measure. An application might be functionally perfect, but very clumsy to use. The best way to understand how the customers perceive a company's services is to send out a survey/questionnaire at regular intervals, ideally a few months after the implementation of a new product.

The format of the survey will depend entirely on the nature of each company's business and who their customers are. A customer may be a sales subsidiary who in turn has its end-user customers, large banks, educational institutions, small companies, or individuals.

Everyday users of software packages don't use metrics and concepts to measure the quality of software packages in the same way as developers do. In [Dehnad90], it states that these users are more likely to express their perception of quality by such statements as "you can learn it in less than an hour", "you only need to read the first few chapters of the manual", "it does everything you might need".

Conduct a survey of the main customers, listing the areas of most important interaction between the software supplier and the customer. The sample form overleaf lists many communications elements such as 'we're kept up-to-date with schedules', 'our queries are answered in a timely manner', and 'we are kept informed of progress, problems and solutions throughout the project'. The type of statements that each company will use in its survey will depend on the company's process and its agreed obligations to its customers.

The information gathered from the survey will help pinpoint areas that need most improvement with respect to the customers' perceptions. Statements that consistently receive a 'sometimes' or 'never' response require specific action. These areas in particular will need some meaningful measures applied to them.

Customer
PROCESS FEEDBACK FORM

COMPANY NAME: DATE:	CONTACT NAME:
Please fill out the evaluations below, and return to:	

Please rate the statements below as follows:

4 = ALWAYS
3 = MOSTLY
2 = SOMETIMES
1 = NEVER

PROCESS evaluation

Statement	Rating	Comments
The requirements specs. are received on time, to enable feedback to be given		
The Test plans are received on time, to enable feedback to be given		
We are kept up-to-date with schedules on a fortnightly basis		
The agreed release dates are met		
Our queries are answered in a timely manner		
Our input is taken into consideration while developing the product		
Beta copies of the software are received as agreed		
Feedback on beta versions is acted upon		
We are kept informed of progress, problems and solutions throughout the project		
Overall, the service we receive throughout the development of products meets our requirements		

COMMENTS:

3.3.2 Staff view.

Software Quality means different things to different people. To a programmer, the modularity of a program's source code might be important, whereas a Project Manager might view productivity as a higher priority. If a metrics system is to be introduced into a company, the only way to get it to work is to directly involve the staff, getting their views of current processes, and how important these processes should be in order to produce quality software.

One reason for conducting a process survey is to find out where the process is broken most (this is a bottom-up approach to metrics implementation). Another very important reason is to demonstrate that it is the project that is being measured, not the individual programmer. If metrics are used to evaluate a single programmer, that person is encouraged to work for the numbers instead of the quality of the system. In the Mythical Man Month [Brooks75], it mentions that on the OS/360 project, productivities in the range of 600 to 800 debugged instructions per man-year were achieved by control program groups, and 2,000 to 3,000 debugged instructions per man-year by language translator groups. The problem with using program size as the basis for productivity is that the same function can be written efficiently in 10 lines of code, or could take 50 lines of sloppy code. The 50 lines of code gives higher productivity, and therein lies the danger of programmers working for numbers rather than the quality of the output.

Karen Hooten, [Hooten92] uses Isaac Asimov's novels to further illustrate this point. In his foundation series, there is a mathematical model that predicts the future. The guardians of the model are careful not to try to apply the model to anything smaller than an entire society. They further guard the secret so that the knowledge of the plan doesn't affect the outcome. The same cautions should be used when implementing a metrics system.

To glean the information from the survey overleaf, some calculation is required when the completed forms have been returned. Firstly, add up all responses to each statement for both the 'current practices' and the 'importance' columns, for each function, and calculate the averages. Subtract the average 'current practice' figure from the average 'importance' figure for each statement. The greater the difference, the more urgent the employees view the improvement of the process area represented by the statement. Generate a prioritised list by ordering the statements from the highest calculated difference to the lowest difference.

Employee
PROCESS FEEDBACK FORM

DEPARTMENT:	CURRENT PROJECT (if applicable):
DATE:	JOB FUNCTION:
Please fill out the evaluations below, and return to:	

Please rate the statements below as follows:

Rate the current practices in your experience

4 = Strongly Agree
3 = Agree
2 = Disagree
1 = Strongly Disagree

How important are these practices to our success, in your opinion?

4=Essential
3=Quite Important
2=Not Important
1=No relevance

PROCESS evaluation

Statement	Current Practices	Importance in your Opinion
Project Planning plays an important role from the start of the project		
Quality planning plays an important role from the start of the project		
Schedules are realistic and kept updated		
There are enough resources (people and machines) to enable project completion on schedule.		
Risk assessment is used throughout the Development process		
Post Mortems are used to learn from one project to the next, and across projects		
Customer involvement is ongoing throughout the process		
There is a concentrated effort to ensure that errors are not introduced during the Development process, rather than spending a lot of effort on testing for and correcting errors later on.		
Standard methods and procedures are used throughout the Development process		
Standard tools are used across projects		
Training in the process is timely		
Tools training is timely		
Process measurements are recorded and play an important role in identifying areas for improvement		

COMMENTS:

3.4. Define applicable metrics categories

The results from the questionnaires above, along with the key process areas information from the Capability Maturity Model will help focus on areas that need particular attention, and should point to metrics categories that would most benefit the process improvement effort.

> "*A company can be considered as an imperfect bucket that is used to take water out of a well. A certain amount leaks. Some water slops over the top. Because of all these losses, the water carrier has to devote additional effort to lifting water. In other words, he lifts considerably more water than the value of the water achieved. Stopping up the holes in the bucket is not enough. The user has to understand how the holes came into the bucket in the first place, and take sufficient steps to ensure that they do not reappear elsewhere.*" [Choppin91]

Part of the understanding of the holes in the 'bucket' is to be able to identify them, measure their size, and measure the amount of water that is leaking out. Translating this to a company's process, it is to identify the process areas that are most in need of repair, and to introduce measures that will tell both the extent of the problem and the cost of these problems to the company. These measures must also be able to demonstrate improvement as time progresses.

An exhaustive list of metrics will have such an overhead to implement, that success would not be likely. Therefore, it is best to start small, proving the value of process metrics, and increase the number and type of metrics as time goes by. Trying to do everything at once is impossible, so an evolutionary approach should be taken. Start simple, with basic measures, and work towards more detailed measures as the process matures.

For suitable measures to be chosen for the company, the applicable metrics categories need to be understood. The next section describes the broad categories that could be implemented, and section 3.6 describes specific measures to various levels of granularity that can be implemented, depending on the current maturity level of the process.

3.4.1 Metrics Categories

Suggested metrics categories, which are described in the following pages, are:

- Size
- Rework
- Schedule
- Productivity
- Effort
- Quality

Which measurement categories to concentrate on, and the granularity of measures depends on the capability maturity level of the development process, and on the development life-cycle itself. Each company must decide for itself what to include for the initial measurement programme - the metrics described in chapters 4 and 5 are primarily suitable for a level 2 company, with a subset of these suitable for a level 1 company, and a superset suitable for level 3 projects. Metrics are expensive to implement, in terms of the overhead involved in capturing the data on a regular basis, and providing meaningful reports. Therefore the fewer metrics a company choses the easier and the less expensive it is to implement them. However, having too few metrics might not work either. The key is to have the right measures for the right areas.

The Software Engineering Institute also recommend a list of metrics categories to be used in conjunction with the Capability Maturity Model [SEI92-TR25]. They choose the following metrics categories, which look different to the list above, but are in fact fairly similar. Areas important to DoD systems, but not quite as important to the 'average' software development environment, such as stability, are not included in the above list. That list contains the important categories for implementing a metrics system with a primary focus on process improvement.

- Progress
- Effort
- Cost
- Quality
- S/W QA audit results
  - review results
  - trouble reports
  - peer review results
  - defect prevention
- Stability
  - requirements stability
  - size stability
  - process stability
- Computer Resource Utilisation
- Training

a) Size Metrics

Size metrics are normally expressed in terms of physical source lines and logical source statements. They are used to help planning, tracking and estimating of software projects. They are also very useful in computing productivity metrics, to normalise quality indicators, and to derive measures for memory utilisation and test coverage. The size of a software project is commonly expressed in term of Lines Of Code. However, this can be somewhat ambiguous - should comments be included? should declarations? compiler directives?. The Software Engineering Institute of Carnegie Mellon University has published a framework for constructing and communicating definitions of size [SEI92-TR20]. Included are checklists for defining and describing the coverage of Source Lines of Code and logical source statements. The checklists allow companies to include or exclude an exhaustive list of elements in the definitions.

Lines Of Code is the simplest metric, but as explained in section 3.3.2, the number of lines of code might not be a very relevant metric, as what one programmer writes in 50 lines of code, may take another programmer only 10 lines of code to write. Boris Beizer [Beizer84] suggests that using lines of code to measure complexity is no more scientific than weighing the listing, or measuring it with a ruler (assuming that all paper is supplied by the same vendor and all listings are done under the same operating system). My belief is that lines of code can be a fairly useful measure if there is some consistency in coding standards across projects.

Other measures relating to size, and very useful at the project planning phase are metrics used to aid estimation, such as complexity metrics and function point analysis. Using function points instead of lines of code leads to better, more accurate estimates of how long a program will take to develop, based on its complexity. Lines of Code are not appropriate to a 4GL environment, whereas function points are suitable. Complexity measures will be discussed with productivity metrics in the next section.

An alternative to using lines of code as a basis for measures is to use Function Point Analysis, developed by Allan Albrecht in IBM, 1979. Function Point Analysis includes three factors that affect the end result - information processing size, technical complexity and environmental factors. You assign a value (function count) based on the amount of information processed and provided by the resultant system (no. of input fields, no. logical files, no. of output fields, no. files accessed, and no. online inquiries). That value is multiplied by a numeric rating of the technical complexity of the project (the weighting factors are according to three complexity types - simple, average or complex). Finally, this value (total unadjusted Function Points) is multiplied by a numeric rating of 14 environmental factors (eg transaction rate, installation ease, end-user efficiency, etc.). A full description on how to calculate function points is in section 4.1 e).

Function points measure the units of work a program actually delivers to end users, thereby avoiding the shortcomings associated with metrics based on lines of code. Albrecht [Albrecht83] gives three major reasons for using function points as a measure:

- The function points count can be developed relatively easily in discussions with the customer/user at an early stage of the development process
- It makes needed information available - since a statement of basic requirements includes an itemisation of the inputs and outputs to be used and provided by the application from the user's external view, an estimate may be validated early in the development cycle using function points.
- Function points can be used to develop a general measure of development productivity, eg function points per person-month, or person-hours per function point.

My view is that using Function Points can aid communication with customers, as their requirements can be translated into numbers of function points, which is a tangible entity. Customers can then get a better appreciation of the demands they are placing on the developers. Function points is best suited to data-intensive systems, with low procedural complexity.

b) Productivity:

Productivity metrics are normally expressed in terms of quantity of work per person-day or person-month on the project. These can help answer questions such as 'has changing over to a 4GL environment made things better?' Process improvement efforts normally have a positive effect on productivity rates, in conjunction with improvements in other areas, such as quality and cost.

Productivity measures on their own can be too simplified, and only come into real significance when used in conjunction with effort metrics and staffing levels. Robert Green [Green92] compares Western software development projects with dam building projects in China, that is managers throw lots of people at the projects in the hope that productivity will rise and the projects will finish on time. There is a minimum number of programmers required to build a system, but it is not true that an increase in programmers will aid productivity. There is a point in most software development projects where the addition of staff will reduce productivity. Brooks' Law [Brooks75] states:

Adding manpower to a late software project makes it later

The Project Manager realises that the proposed schedule will not be met at current progress rates, and therefore decides to add more staff to try to alleviate the problem. Adding staff increases the number of communication paths within a team, which results in more breakdowns and lapses in communication. Brooks [Brooks75], says that if there are n workers on a project, there are $n(n - 1)/2$ interfaces across which there may be communication, which means that three people require three times as much intercommunication as two, and four people require six times as much as two.

As an example, assume a product of an estimated 20,000 Lines of Code. Take a productivity rate of 20 LOC per programmer-day. This gives an estimated effort of 1000 person-days, or 5 person-years. If productivity and effort were a matter of simple mathematics, then 2 people over a period of 2.5 years, or 5 people for one year or 10 people for 6 months would get the job done equally well. If this were true, it could be extrapolated even further to say that it would take 20 people 3 months to complete the same project. Common sense says that this simply won't work, although more people are often thrown at a project to try to get it out of trouble. Quoting Brooks once more [Brooks75], he states that one cannot get

workable schedules using more people and fewer months. He says that the number of months of a project depends upon its sequential constraints. The maximum number of people depends upon the number of independent subtasks.

Boehm considers it impossible to compress the schedule below 75% of the nominal schedule, which is defined by the COCOMO schedule (TDEV) equations [Boehm81]. The COCOMO models are described in more detail in section 4.1 (d), so the basic equations are used here without explanation.

Using the Basic COCOMO equations (organic mode) on the simple example above, gives:

$$MM = 2.4(KDSI)^{1.05} = 55.75$$
$$TDEV = 2.5\ (MM)^{0.38} = 11.52$$

MM/TDEV gives an average staffing of 5 people (Full-time-equivalent Software Personnel).

If we want to reduce the schedule by the maximum, ie to 75% of the original estimated schedule, we're setting TDEV = 8.64. Boehm gives the development Schedule Constraint overall Effort Multiplier (SCED) for maximum schedule acceleration as 1.23, which gives effort (MM) = 68.57. Average staffing for this scenario is therefore 8 people.

The above example gives a nominal figure of 5 people for eleven-and-a half months, or for an accelerated schedule, the figure of 8 people for just over eight-and-a half-months. In this example, reducing the schedule by 25% implies increasing the personnel by 60%. Further schedule compression is not possible.

By improving the process, productivity rates may increase. This can be due to new tools or alternative methods. As an example, if a company decides to adopt a re-use philosophy, the amount of time necessary to develop a product using previously implemented and thoroughly tested code segments will be less than the time necessary to design and implement the features from scratch. Another company might move to a 4GL development environment, which should show improvements in productivity rates for new projects.

c) Rework Metrics

Rework metrics can be expressed in % of total effort, % of total work completed, or % of the project cost. A useful metric might also be number of rework hours per thousand Lines of Code. Phil Crosby, on his Quality courses, states that in service industries, at any time, up to 40% of the people are re-doing things that should have been done right in the first place.

Rework metrics answer questions such as 'what is it costing us to re-do things?' 'What processes give us the worst rework rates?', and 'where should we concentrate our improvement efforts to get the most benefit in the least time?'

Sometimes rework metrics can be very difficult to calculate, as the rework may be due to several different causes - one being mistakes made by the developers themselves, another being change requests from the customers. Issues that need to be addressed by any organisation before measuring rework is what is to be included in the rework metrics category, and if there are several categories, what level of granularity makes the most sense?

Rework metrics require particular caution when implementing them, as there is a very strong temptation to use them as a measurement of individual programmer performance. One must not succumb to this temptation, and has the additional task of assuring the programmers and other individuals that the rework metrics are for improving the development process, and are never going to be used to hold individuals to ransom. A culture should exist in the organisation, which encourages people to raise issues as early as possible in the process, rather than covering up mistakes. A mistake costs magnitudes more if found and corrected at the testing stage. Never shoot the messenger.

Rework measures provide an excellent insight for process improvement efforts. If a 'right-first-time' approach is taken to development, then the amount of time spent on rework should decrease. 'Right-first-time' in the context of software development is somewhat different to its perception in manufacturing. For software development, it means adopting the attitude of each person checking their own work thoroughly before declaring that it is complete. This approach should be adopted by all project personnel, from requirements definition right through to implementation of the completed system.

d) Effort and Schedule Metrics

Effort metrics are of two kinds. The first relates to the early phases of the project, and help in project planning, eg. Boehm's COCOMO metrics, Halstead's Software Science Metrics and McCabe's Cyclomatic Complexity Metric. These metrics are also considered as size metrics.

The second type of effort metric relates to tracking effort as the project progresses, and relating this to what was planned.

Effort and schedule metrics are expressed in terms of % time, person hours, and activities completed against what was planned. They help in estimating the cost of the project. They quickly point to areas where corrective action is needed. The Software Engineering Institute has published a framework report on schedule and effort measurement [SEI92-TR21], which contains checklists for defining staff-hours, and defining which tasks are included in the measurements.

Effort and schedule measurements answer questions such as 'will the project be completed on schedule, and if not, when will it be completed?' and 'will the project be completed within the planned amount of effort, and if not, how much effort will be needed? In terms of process improvement, the actual vs. planned effort and schedule measures should show an improvement in correlation over multiple project releases. If the process is under control, initial project estimates should be fairly accurate. As the development process improves, the effort required, and the related project costs will be reduced.

Effort metrics display expended resources over time, which shows current status, and can forecast actual effort expended at completion. They show actual versus planned person-hours expended for the current and past time periods - a reasonable time period to use would be by month.

Schedule metrics, also known as progress metrics, use measures of work scope to track and show the progress toward completing activities, tasks and work packages. (These metrics are explained in detail in section 4.2.1). Schedule metrics provide a quantitative basis for managing a project by reflecting actual schedule progress against planned schedule progress for current and past time periods.

e) Quality Metrics

Quality metrics answer questions such as: 'How stable is the product?'; 'How many operational faults were found?'; 'How effective is our test strategy?'. Probably the most important question they help answer is 'How effective is our development process? Quality measures can be expressed in terms of defects per thousand lines of code, defect find rate, paths that have been tested, code coverage, etc.

Quality metrics relate to the number and type of defects in the product. They are the basis for defect prevention techniques. Defect data for each project should be kept in a database, from which the relevant metrics can be calculated. Analysing the data related to defects:

- Helps determine when to stop testing the system and release it
- Identifies error prone modules and development activities
- Helps assess the effectiveness of testing and development techniques.

To improve the process, the Quality measures will provide the best indicator of areas that require most immediate attention. It is also easy to compare projects and demonstrate improvement from one product version to the next using quality measures. Quality Metrics are presented in detail in Chapter 5 - applicable quality sub-categories are briefly mentioned here.

Firstly, there are the models used for planning purposes, to predict how defect-prone the modules are before commencing testing, and encourage individuals to spend more time testing those modules that are most error-prone, and/or enable them to stop testing when a certain level of defect density has been reached. These measures include Halstead's bug prediction formula, McCabe's complexity metric and Musa's failure intensity objective.

The second quality sub-category is measuring defects as testing progresses in terms of type and severity, to help pinpoint problem areas, and provide historical data. Other measures used here would be defect resolution per module, lifetime of active defects, and defect rate with respect to program execution time, ie mean-time between failure.

The final category comes into effect when the software has been released. This category includes release measures such as post-release defect counts, number of product re-releases, and analysis of the total bugs reported for the project to produce pareto charts (ie 80% of the bugs are in 20% of the modules, and 80% of the bugs are from 20% of the causes). This information is then fed back into the process in order to improve for the next project.

The following defect data categories are commonly used:

- *Defect Count* is a simple measure of the number of defects found in the product, reported at the end of the project.
- *Defect Find Rate* shows the number of defects found per time period. The number of new bugs found in each time period shows how stable the product is, and indicates whether or not the team are close to producing a release candidate.
- *Defect Distribution* shows the distribution of defects per module or per phase of development.
- *Defect Density* is the number of defects per KLOC, (function points can be used instead of Lines of Code). It is very useful for comparing defect ratios across different projects.

3.4.2 Level-specific metrics

Section 2.4 gives a tabular summary of the types of metrics to implement at each process level, based on some research done by the Contel Technology Centre [Pfleeger90]. In Section 3.4.1, the suggested measurement indicators, ie categories, are listed. This section expands on what was described in these two previous sections, with my interpretation of the metrics suitable for each of the levels 1 to 3, followed by a table summarising the applicable measures (each of these measures are explained in chapters 4 and 5). The metrics evolution process in going from level 1 to 3 is noticeable here - the same categories are used, but the metrics become more specific and granular.

a) For level 1 organisations, the important measures are those that will demonstrate when software projects are beginning to come under control. These would be defect counts, basic productivity measures, basic effort measures, and basic size measures. The SEI does not include level 1 metrics in their report. The measures for the initial implementation in a level 1 company are those collated at the end of the project, ie to perform full project post-mortems. The aim is to learn from one project to the next, and to compare across projects.

b) For an organisation at level 2, the metrics will become more granular in each of the categories. These measures include actual vs. planned cost, schedule progress, defect metrics, complexity metrics, etc. The key here is being able to measure actual vs. planned, and to be able to introduce successful countermeasures where results are not as planned.

c) An organisation at level 3 would have a fairly good metrics system in place, and would work on further refining it. The granularity of the measures and the number of measures will be greater than previously. A full set of project metrics would be implemented at this stage, and measures would be analysed throughout the product lifecycle. One of the key issues here is to show distribution of the metrics over a range of values, and to be able to act on the information, implementing counter-measures, as the project progresses, ie as early in the project lifecycle as possible. The cost of implementing and sustaining such a metrics system is very high, so the concentration here will also have to be on automating the metrics collection and analysis processes.

Category	*Level 1*	*Level 2*	*Level 3*
Size/ Complexity	Lines of Code	*Choice of:* Lines of Code Halstead's tokens McCabe's Cyclomatic Complexity De Marco's Bang Metric Function Points	Choice is same as Level 2
Productivity	Productivity rates per function - end of project	Actual vs. planned productivity per function per phase	Actual vs. planned per function/phase within approved range, based on work packages complete
Rework	Total rework quantity - end of project	Rework quantity and effort expressed as a % of total	Rework effort and quantity per cause category and as % of total
Effort	Total effort per function - end of project	COCOMO Per function per area (actual vs. Planned) Non-cumulative Effort distribution (monthly)	Automated effort tracking system Cumulative Effort Distribution
Schedule	Simple Gantt chart	Gantt chart with % complete, on monthly basis	Progress curves categorisation throughout project Cumulative Work Packages Complete
Quality	Total defects Defect density	Defect find rate profile by week Defect density Defect severity Pareto analysis of type	Musa's software reliability measures Defect find rate profile (by day) Defect density Defect severity Pareto analysis by cause Age of defects

3.5. Break corporate goal into a specific goal for each category

The overall corporate goal for improvement needs to be broken down into further goals. Section 3.2 describes how the overall corporate improvement goal is defined, using the Goal-Question-Measure approach. At this point, the goal can be broken down into goals of further granularity, ie from the corporate goal to goals of Managers and to project team goals.

These goals would be defined in terms of each metrics category, so that progress towards them can easily be monitored. The goals would be further broken into more specific goals for the projects in terms of improvement targets. Each function on the project team would have their performance goals expressed in terms of metrics that were directly related to them. If this is successful, there would be an objectives chain from the top of the company to the bottom. These objectives/goals would serve as the basis for day-to-day improvement.

Section 3.2 above states that the approach adopted by the Metkit consortium [Metkit92a], defines goals in terms of

- what is wanted
- who wants it
- why it is wanted
- when it is wanted

Taking each metrics category, a company should come up with a corporate goal that clearly states the expected outcome and the timeframe. Taking the rework category as an example, one might state the goal as:

To reduce the coding rework as a result of programming defects by 50% in 18 months.

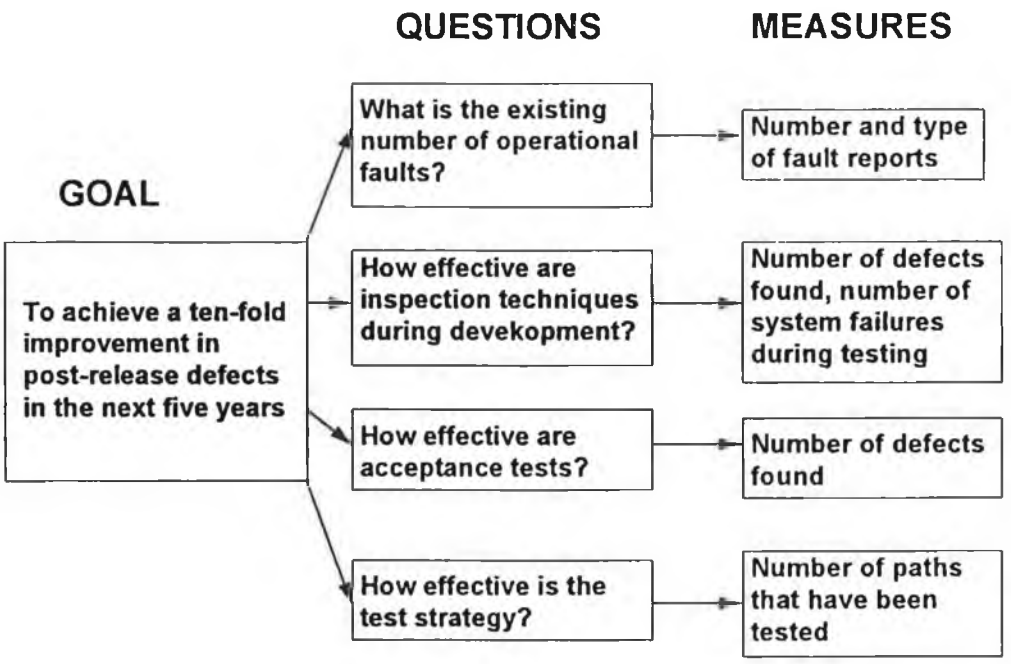
or

To reduce the amount of rework, due to changes in customer requirements within 1 month of the expected delivery date, to zero within a year.

Knowing the goals, questions can then be asked relating to each goal, in order to identify the specific metrics to be defined and implemented in the system.

A software reliability goal is used to demonstrate how specific measures can be selected via the Goal-Question-Measure approach, as described in [Ashley91]

The GOAL-QUESTION-MEASURE Approach



3.6. Define specific measures

This is done, based on the goals established and the questions that have resulted from these goals. The 6 metrics categories: Size, Productivity, Rework, Effort, Schedule, and Quality should also be kept in mind. The metrics definitions must be clear, concise, unambiguous and understood by all involved. Two people taking the same measures on the same items should get the same results - this is known as interrater reliability. And secondly, two different methods used to determine the same metric on the same items should also get the same results, which is known as intermethod reliability. In order to satisfy these criteria, metrics must be very precisely defined. Chapters 4 and 5 detail specific measures that can be implemented in each metrics category.

The Mermaid Project, [Mermaid91], identifies 11 principles for determining metrics. The six that are most relevant to the process measures described are:

1. Define clearly what the metric is supposed to measure, ensuring that different issues are not raised in the same metric.
2. The process of metrication should be as objective as possible
3. Natural language is inherently ambiguous. Extreme care must be taken to minimise the likelihood of the metrics definitions being misunderstood. eg. avoid using 'usually', 'fairly', 'likely', 'often'.
4. Where external standards or formalised methods are being referred to, it must be ensured that all those who will be using the metric know to what/whom the standard and/or formalised methods refer.
5. Where possible, avoid making comparative assumptions about knowledge of the requirements of previous projects.
6. Once the metrics are in use, the response patterns should be examined regularly for intrinsic error:

   If there are a number of respondents, then the pattern of one individual's response which is consistently different to that of his colleagues, may be due to two reasons. Firstly, his projects may be markedly different; secondly, he may be intentionally optimising the values in order to improve his position within the organisation.

3.7. Prepare Data Sheets

Data sheets need to be prepared once the important metrics categories, for the organisation to implement, have been identified. These should be intuitive, easy to understand and fill in, with the responsibility of who is to fill what in, clearly stated and understood by all involved. Ultimately, the data should be entered directly into a database, which would generate the appropriate metrics reports on request.

However, since a company should always plan to throw one away [Brooks75], the chosen metrics should be implemented manually, on a three-month basis, and then reviewed. Some will prove more useful than others, some will take too much time to gather the data, and other will need some adjustment. Only then should the system be automated. Otherwise, the rework rate involved in amending the automated system will be too high.

The following is an example of a data sheet for effort and defect counts (Add-ins are executables included in the product, along with the main executable).

Team Leader Name:		Project Name:	
Date:		Reporting Period:	

Effort

Component ID	*Testing Effort*	*Defect Removal Effort*
Main Executable		
Help File		
Install Program		
Add-ins		

Defect Counts

Component ID	*Requirements Specification*	*Design*	*Code*	*User Documentation*
Main Executable				
Help File				
Install Program				
Add-ins				

3.8. Provide necessary training

The objective of metrics training is to promote the understanding and use of metrics so that all employees take ownership for the implementation of metrics on their project. The amount and level of training each employee should receive depends on their function and their responsibility for metrics collection and reporting.

A training plan should be drawn up to encompass the differing needs of each function. All employees should understand the basic principles of measurement, the categories of measurement that will be implemented in the organisation, and their usefulness in helping to understand, control and improve the development process. Managers need to understand the role of measurement in helping them to manage their projects, and they need to understand the costs and benefits of implementing a metrics system. It is also likely that new methods and tools will be introduced for metrics determination, collection and reporting, which will require training for specific functions. Examples of such new methods could be Function Points Analysis, Pareto analysis, Software Reliability Measurement, etc.

A training grid would be a useful aid in identifying who should receive what training. This would be a spreadsheet, with the columns representing each staff function, and the rows representing each training module, a sample section of the grid could be as follows:

Module	S/W Manager	S/W Dev. Engineer	S/W Test Engineer
Metrics costs and benefits	√		
Metrics Principles	√	√	√
Metrics Categories	√	√	√
Size Measures		√	
Quality Measures		√	√
etc.			

Instead of each company developing its own training modules, they could use training modules that have been developed by the MetKit Consortium. MetKit is an Esprit II project (no. 2348), sponsored by the CEC, the result of which is a series of training modules for both education and industry. The Industrial package consists of a total of 18 modules. Each module pack consists of all materials required to deliver the course in-house (including slides, teacher notes, student notes, and Questions and Answers)

The modules are:

1. Measurement As A Management Tool
2. Introduction To Software Engineering Measurement
3. What Is Measurement?
4. Procuring Software Systems
5. What Can We Measure In Software Engineering And How?
6. Estimating The Cost Of Software Development
7. Establishing A Cost Estimation Measurement Programme
8. Cost Estimation Strategy
9. The Case For A Standard Work Breakdown Structure
10. Principles Of Function Point Analysis
11. Process Benchmarking
12. Process Optimisation Measures
13. Specifying And Measuring Software Quality
14. Usability Assessment
15. Defect Analysis As An Improvement Tool
16. How To Implement A Measurement Program
17. Case Study - Setting Up A Measurement Programme
18. Software Engineering Measurement In Industry

3.9. Measure current processes and products

In order to set realistic targets for short-term improvements, the company needs to know where they are now, with respect to each of the goals and metrics. Imagine some people from a company have been brought to a place, which could be anywhere in the world, from where they need to travel to somewhere in Ireland within a week, ie they have a general idea of their ultimate destination and the timeframe. They have also been supplied with a map and money, ie a way of getting there. Now, in order to make it to Ireland, they would like to split the journey into several legs, say to cover a certain distance each day, ie split the journey into manageable portions. If they don't know where they currently are, having a map and knowing the destination will be of no use to them. Therefore, before setting out on the journey, they need to find out where they currently are. At this stage, they only have a general idea of their destination - section 3.10 describes how to define the ultimate destination and to choose the places they should pass through on the way there.

So how does a company find out where they are? The starting point is to measure projects just completed, to establish the baseline, and to answer some of the questions posed by the Goals-Question-Measure approach described in sections 3.2.2 and 3.5. The same measures can be used at the end-of-project (ie to measure past projects) as are used throughout the process, without using time as the basic unit. Use as many of the measures selected for the company measurement process for this exercise.

A very useful exercise to carry out is a Pareto analysis of defects found on previous projects, to help identify areas for improvement, and to give a graphical representation of the current quality of developed software. Pareto analysis is explained in section 5.4.

Current Cost of Quality

A good exercise to complete is to measure the Cost Of Quality as a percentage of overall operating costs. It is next to impossible and would cost too much in terms of time and effort to get a fully accurate cost of quality, but a good approximation is all that would be required for the purpose of implementing metrics for process improvement. Managers need to know where the more significant costs are to be found, and then work to severely curtail or eliminate them.

The Quality Costs should be classified into types, from which the areas in which effort should be concentrated to maximise improvements can be identified. Companies have more control over some areas than others (direct costs vs. indirect costs), and should strive to eliminate rework costs, whilst increasing planning and defect prevention activities. Measuring the cost of quality on an annual basis, using the figures received from the implemented metrics system, gives an objective measure of the cumulative effect/benefit of the process improvements efforts.

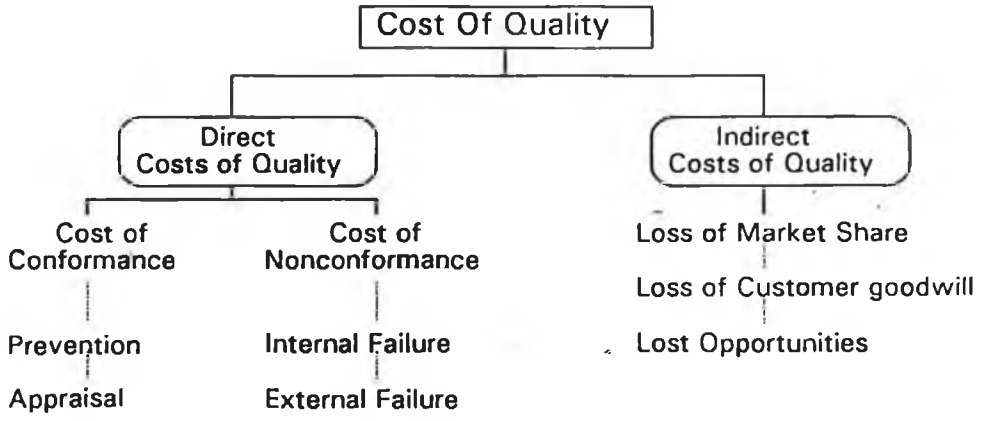
The two main categories of quality costs are direct costs and indirect costs. The direct costs are the easier ones to measure, as the indirect costs are things like loss of market share, lost customer goodwill, cancelled orders, etc., which realistically are rather vague.

Direct costs are comprised of the cost of conformance (i.e. achieving satisfactory results) and the cost of non-conformance (ie dealing with failure - rework costs).

The cost of conformance can be further broken down into prevention costs and appraisal costs. Prevention costs are those that are incurred before the process, thus preventing failure - i.e. planning, training, etc. Appraisal costs are the costs of those activities taking place during and after the process, such as Testing, etc.

The cost of non-conformance is further sub-divided into internal failure costs and external failure costs. Internal failure costs are those that occur before and during release, i.e. bug fixing, regression testing, all rework. External failure costs are those involved in fixing problems after release.

The following diagram gives a graphical representation of the cost of quality calculations. It provides the top-level indicator of the development process, and can be expressed as a percentage of total operating costs, on an annual basis.



3.10. Set improvement targets

If you don't know where you're going, any road will do

- Chinese Proverb

Section 3.9 above describes how to find out where a company is now. This section describes how to set out where they are going. This document as a whole will show them how to get there.

The example from section 3.9, of some company employees being taken to a place which could be anywhere in the world, from where they have to travel to Ireland in a week, is continued. At this stage, they have established where they are, they have a map, and they know the destination is somewhere in Ireland. The next step is to find out where exactly they are going - Dublin? Athlone? Belfast? Tralee? This is important, as they will have to divide the journey into manageable stages, and to get to Tralee could mean a slightly different itinerary, or mode of transport, than required to get to Dublin.

In section 3.5, the Corporate goal is further defined into a specific goal for each metrics category. Taking a combination of the corporate goals and the knowledge of where they are now with respect to the chosen metrics, it should be relatively straight forward to define specific targets for 6 months, 1 year and 2 years' time.

Many organisations have unrealistic expectations when first starting out with a measurement program. If they are not too ambitious, and adopt an evolutionary approach, it should work out.

To tie it all together, take the goals defined in section 3.5 together with the measures selected for the process and take the current process metrics obtained from the projects just completed (section 3.9). Examine the goals in the light of what the measures of the current process show, and state the long-term goals in terms of what these measures should show when these goals have been successfully achieved. From there, take each long-term goal, and express it as a series of short-term goals with specific quantitative targets and a timeframe.

Taking the example of the journey again - the company has to decide exactly where they are going to, in terms of co-ordinates on the map. They already know exactly

where they are, so using the map, and knowing their own abilities and limitations, each leg of the journey can be planned.

3.11. Automate the system

The first implementation of the system will be a manual one (using spreadsheets and documents), as there will be several changes, and it is better to plan to automate the second system, rather than spending the time automating the first system, then having to spend effort reworking the system for the changes - as Brooks says:

Plan to throw one away; you will, anyhow

[Brooks75]

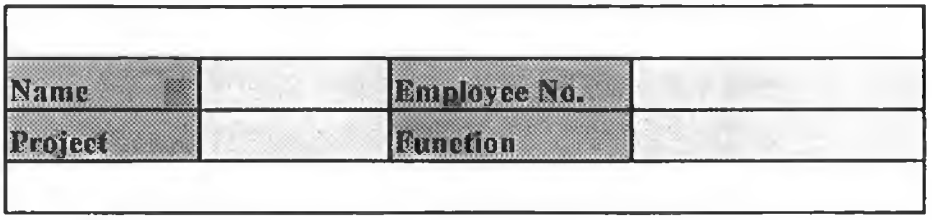
In an automated process-metrics system, there should be, at a minimum, a project management module, an effort-tracking module, a defect-tracking module, and a reports module. The project management module would contain the size, complexity and schedule information. The time tracking module would contain effort information, and would tie-in with the schedule information. The defect-tracking module would contain all defect information, as described in section 5.1. The reports module would contain all the required reports which would be generated by performing the necessary calculations by accessing the data from each of the other modules.

The development of the defect database, described in section 5.1, should be completed first. The project management module should use a project management product, which will be able to produce Gantt charts and PERT networks of the planned vs. the actual schedule. The automation of the effort data collection and metrics reporting modules should take place about three months after the manual implementation of the system. The easiest form of automation is to use a database for data collection. This is particularly suitable for collecting the effort data .

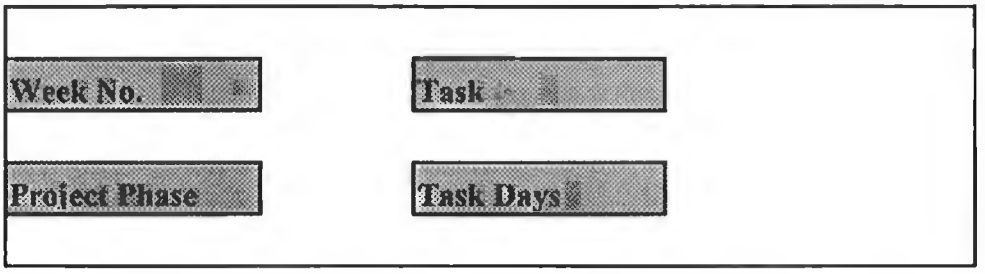
The project management module should contain two types of data - firstly, the size and complexity information should be contained in a template (this could be kept in a database, separate from the schedule information), which would be used as a method of measuring actual vs. planned quantity of work, and will be used as a basis for calculating many of the metrics (eg productivity, defect density); secondly, the development process as specified in section 3.1 should form the basis of the work breakdown structure for the schedule. The system should allow

schedules to be created, maintained by recording % complete data, viewed without making changes, and should produce reports such as a Gantt chart, Pert network, critical path list, etc.

The effort-tracking system tracks actual effort, in man-hours expended on each task. The first screen of the effort-tracking system should contain the employee name, number, project, function, etc. The next form to be displayed should depend on the entry given for Function.



The following is a generic example of an entry form, at its simplest. Each field would have a drop-down list of legal values, depending on the function and project selected on the first screen. *Week number* is the current calendar week number; *Project phase* would contain the phase being worked in; the legal values for *task* would be dependent on the project phase and function, and in the *task hours* field, the number of days spent on that specific task would be entered.



Metrics calculation is a matter of dividing one set of data by another, to produce a result which can be compared across several projects. Many of the metrics described in chapters 4 and 5 require some calculation, either based on time or based on quantity of work completed. The metrics reports generated from the data should be in a standard format, for inter-project comparisons, and the frequency of such reports should be agreed with the teams concerned. Some reports are most useful at project-end, whilst others should be produced every month, or more frequently. The frequency of the reports depends on the size of the project, and the number and usefulness of the metrics to the project team. Reports should only be generated if they are required, and if they will add value to the process at that time.

3.12. Review the effectiveness of the metrics

How can the effectiveness of the measures be ascertained? One way is to use further measures, such as the percentage of 1-year targets that were met within the year. Another method is to ask specific questions within 3 months, 6-9 months and 1-1.5 years of implementing the metrics system.

The questions asked after three months should ascertain which of the measures are too time-consuming to collect vs. the gain obtained from them. The three-month review aims to weed out the non-relevant metrics and provide a concise listing of what will be measured and why. A common tendency of companies is to be overly optimistic at first, and they try to measure too much at the first attempt. A company new to measurement should start with end-of project effort, schedule and quality metrics, and to build on these to include in-process measures and a further granularity of measure after about a year, as outlined in section 3.4.2. The system can be automated after the three-month review, as described in section 3.11.

The 6-9 month evaluation asks - have the metrics identified the correct areas for process improvement? Further adjustments may need to be made at this stage to the metrics or the improvement goals themselves. Different reports may also be required than those implemented after the 3-month metrics review.

After 1 to 1.5 years, some of the benefits should begin to show in improvements to the development process. The questions to be asked at this stage aim to demonstrate the improvements, ie by how much has the process improved? The project post-mortems should provide an objective measure of the process improvements, as the actual vs. planned measures should be provided, and compared against the end-of-project figures for the previous version of the product. At this stage, some further metrics may be required, to measure more processes, or to measure to a finer granularity.

The metrics system should be continually assessed and improved upon. Measurement should become an integral part of the process, just as Hitachi, Toshiba, NEC and Fujitsu have accomplished [Cusumano91]. These four Japanese companies have successfully organised the design and development of large software projects using statistical methods adapted for software from the more established engineering disciplines.

4. Specific Measures to implement

4.1. Specification/Code-based Measures

The measures described under this category all help in the planning process. These measures are used to estimate project size, complexity, time and effort. They provide a means whereby good estimates can be made from the detailed specification or design. As the development process becomes more under control, and more stable, the accuracy of these estimates should increase. A number of measures are described under this category - which one(s) to choose depends on the current development process and methods in use.

a) Halstead's Software Complexity Measure

Halstead considers a program as a collection of tokens (operators or operands) [Halstead77]. The measurements of the program are based on counts of these tokens. He has proposed a number of quantities - those that are described briefly here are the Volume metric, the length estimation, the effort estimation and finally the time estimation. The bug prediction formula will be described briefly in the Quality metrics section.

All of Halstead's metrics are based upon the following parameters:

n_1 = the number of unique operators in the program (eg. keywords)
n_2 = the number of unique operands in the program (eg. data base objects)
N_1 = the total occurrences of operators in the program
N_2 = the total occurrences of operands in the program

His program volume metric, V, is defined as:

$$V = (N_1 + N_2) \log_2 (n_1+n_2)$$ - *The unit of measurement of volume is bits.*

The vocabulary of a program is the number of distinct operators and operands:

$$n = n_1 + n_2$$

The length of a program is the sum of the actual operators and operands:

$$N = N_1 + N_2$$

The program length can also be estimated knowing only the program's vocabulary, before the program is written. If a program is written after a data dictionary already exists, then it should be relatively easy to estimate the keywords used (n_1) and the data base objects referenced (n_2).

The length's estimator, $\tilde{N}$, is defined as:

$$\tilde{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

An example, from [Beizer84], states that the validity of the relation of the estimated length, $\tilde{N}$, to the actual length, N, has been experimentally confirmed over a wide range of programs and languages. The example given is:

If a program is written using 20 keywords out of a total of 200 in the language, and it references 30 objects in the database, then its length should be $20\log_2 20 + 30\log_2 30 = 233.6$, which Beizer states is very close to the actual length measured on the program.

Another one of Halstead's quantities is the Effort metric, E, which is defined as:

$$E = N n_1 N_2 (\log_2 n)/(2n_2)$$

since $V = N\log_2 n$, the above equation can be simplified to:

$$E = (n_1 N_2 V)/(2n_2)$$

T, Time Estimation, is:

$$T = E/S, \text{ where S is approximately 18}$$

Vincent Shen, [Shen83], concludes that the 'real-world' use of Halstead's software science measures must be done very carefully. One of the difficulties cited is that the very base of software science (counting operators and operands) is weak, due to ambiguities concerning what should be counted and how. He states that serious difficulties have been the failure to consider declarations and input/output statements and (possibly) counting a "GO TO label" as a unique operator for each unique label.

My view is that just counting the operators and operands in the program (ie n_1, n_2, N_1, N_2) should suffice as a good indicator of size, from which comparisons can be made across projects, and rough rule-of-thumb estimates can be made for effort and time estimates, without having to perform detailed calculations.

b) McCabe's Cyclomatic Complexity Metric

McCabe's cyclomatic complexity metric [McCabe76] is described in relation to graph theory, and is based on the number of decisions in a module (it equals the number of decisions in a module, plus one). The cyclomatic complexity is independent of a module's physical size - adding or subtracting functional statements leaves complexity unchanged. Also, basing complexity solely on the number of paths in a module can be misleading, as nested IF-THEN statements lead to an exponential increase in the number of possible paths.

The cyclomatic complexity metric quantifies a basic number of paths in the program, that have the following properties [Ward89]:

1. They visit each node in a graph of the program and they visit every edge in the graph
2. When taken together, the basic paths can generate all possible paths in the program

McCabe [McCabe76] defines the cyclomatic complexity number of a graph as :

$$V(G) = e - n + 2p$$

where:

e = the number of edges
n = the number of nodes
p = the number of connected components

(A connected component is a code module (function or procedure) from start to end)

However, B. Henderson Sellers [Sellers92] states that the above equation only holds true for p=1, and that the cyclomatic complexity of a modularised program with p>1, is defined as:

$$V(G) = e - n + p + 1$$

In McCabe's equation, the cyclomatic complexity number increases with modularisation, so that a program written as three separate modules has a higher complexity than the same program unmodularised. McCabe gives the complexity of a collection of control graphs as the sum of their individual complexities - ie if module A has a complexity of 6 (five decisions) and module B has a complexity of 7 (six decisions), their sum has a complexity of 13. Assuming p is the number of connected components, and i = 1 to p, McCabe gives the complexity of the collection of components, V(G) as:

$$V(G) = \sum_{1}^{p} V(G_i)$$

Sellers ascertains that in modularisation, the number of decisions remains unaltered, and since the cyclomatic complexity should always reflect the number of decisions plus one, there should be no difference between the complexity of the modularised and non-modularised program. The complexity of a collection of control graphs given by Sellers is the sum of their individual complexities, plus one minus the number of connected components.

$$V(G) = \sum_{1}^{p} V(G_i) + 1 - p$$

hence, he concludes that the sum of the parts exceeds the system value by (p- 1).

Tolerances for the Cyclomatic Complexity metric are given in [Henry90]. A V(g) of up to 10 is acceptable (safe zone), from 11-20 should raise a flag - i.e. the additional complexity should be verified to be manageable and/or justified, and a V(g) of over 20 should sound an alarm.

The uses as expressed in [Ward89] are:

- Automatic identification of potentially faulty software before actual testing is started
- Automatic identification of code modules that could benefit from code inspections
- Automated generation of test case data for all software modules
- Well-defined coding standards accepted throughout the lab
- Effective code defect prevention strategies based on restructuring of overly complex code.

c) DeMarco's Bang Metric

[DeMarco82] says that the highly structured specification model of a project describes the requirement of that project. A quantitative analysis of the model will provide a measure of the true function to be delivered as perceived by the user. This is what he terms *Bang,* which *is a measure of total function delivered by the project per dollar invested from project beginning until the system is retired.* Bang is an implementation-independent indication of system size. The central hypothesis is that the information content of the coded system is a well-behaved function of the specification. Needless to say, the metrics derived from the specification model are only as good as the model itself.

Using a highly structured specification modeling standard, and ensuring there is no single redundant statement in the entire set of lowest-level model components, ensures that the size (information content) of the model is a direct measure of usable system function to be delivered (ie gives a direct measure of Bang).

To calculate Bang, the specification model is firstly broken down into a number of primitives - elements that cannot be further subdivided. The Specification model, which contains the function model, plus the retained data model, plus the state transition model, is successively divided and sub-divided until the primitive level is reached. There are six types of primitive which may result from these partitioning activities:

Partitioning of the function model leads to functional primitives and data elements. Each functional primitive represents an undivided element of user policy governing transformation of input data into output data at one node of the network (Data Flow Diagram). Data elements are indivisible numbers, strings and discrete variables (contained in the data dictionary)

Partitioning of the retained data model leads to objects and relationships, and partitioning of the state transition model gives states and transitions.

These primitives are known as p-counts - there are 12 essential p-counts, of which four are most useful in calculating Bang: FP, the count of functional primitives lying inside the man-machine boundary; OB, the count of objects in the retained data model; TC_i, the count of data tokens (data item that need not be subdivided within the primitive) around the boundary of the *i*th functional primitive (evaluated

for each primitive), and RE_i, the count of relationships involving the *i*th object of the retained data model (evaluated for each object).

There are two ways to calculate Bang, depending on whether the system is function-strong, or data strong. The base measure for function strong systems is the count of functional primitives (FP), whereas the base measure for data strong systems is the count of objects in the database(OB). Since some functions and objects cost more to implement (in terms of complexity and size) than others, there are some weighting factors involved in each formula.

To calculate Bang for function-strong systems:
Compute the Token Count around the boundary of each functional primitive. Use this count to look up the table for size correction of functional primitives, to get a value for the Corrected FP Increment (CFPI). For each functional primitive, allocate it to a class (eg simple update, edit, display, etc.). There is a table provided for weightings according to class of function. Calculating Bang is then a matter of the total sum of the product of CFPI and the complexity weighting for each functional primitive. ie, for $i = 1$ to FP:

$$\text{Bang} = \Sigma(\text{CFPI}_i * \text{complexity weighting}_i)$$

To calculate Bang for data-strong systems:
Compute the relationship count involving each object. Use this count to look up the table for the relation rating of objects, to get a value for Corrected OB Increment (COBI). Bang is the sum of the corrected OB increments over all objects. ie, for $i = 1$ to OB:

$$\text{Bang} = \Sigma \text{COBI}_i$$

The main uses of bang are that it is used as an early, strong indicator of effort, and helps project development costs. Another use is the Project Bang Per Buck metric. As project performance improves, this number will increase. De Marco recommends collecting data on a number of projects to establish standards for Bang Per Buck (BPB) performance. This figure can then be used to set goals for the process improvement efforts. Parts of the BPB that are in the future can be predicted, and this information used to keep project members aware of how well they are performing as the project progresses.

d) COnstructive COst MOdel

Boehm's COCOMO metrics [Boehm81] are widely used for size, effort and schedule estimation. There are three models - Basic, giving ball-park figures; Intermediate, and Detailed. The three development modes are organic: small to medium-sized projects in a familiar in-house environment; Embedded: ambitious and tightly constrained projects; and semi-detached: between organic and embedded.

$$\text{Effort} = a\,(\text{size})^b \times \text{product of cost drivers}$$

where there is a list of cost drivers for the intermediate and detailed models, and the values of a and b depend on the mode of development.

There are fifteen cost drivers used in the intermediate and detailed COCOMO models which are grouped into the four categories of software product attributes, computer attributes, personnel attributes, and project attributes, as follows:

- *Software Product Attributes*

RELY Required Software Reliability
DATA Data Base Size
CPLX Product Complexity

- *Computer Attributes*

TIME Execution Time Constraint
STOR Main Storage Constraint
VIRT Virtual Machine Volatility
TURN Computer Turnaround Time

- Personnel Attributes

ACAP Analyst Capability
AEXP Applications Experience
PCAP Programmer Capability
VEXP Virtual Machine Experience
LEXP Programming Language Experience

- Project Attributes

MODP Modern Programming Practices
TOOL Use of Software Tools
SCED Required Development Schedule

The basic COCOMO Model for effort estimation (Organic mode) is

$$MM = 2.4(KDSI)^{1.05}$$
$$TDEV = 2.5\ (MM)^{0.38}$$

Where:
MM = Manmonths,
KDSI = Thousands of Delivered Source Instructions,
TDEV = Tome for Development

Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is necessarily limited because of its lack of cost factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes known to have a significant influence on software costs. Both the intermediate and detailed models cater for these cost factors.

The Intermediate model is effective for most cost-estimation purposes, but has two main limitations when it comes to detailed cost estimations for large projects; its estimated distribution of effort by phase may be inaccurate, and it can be very cumbersome to use on a product with many components (because of the separate cost driver ratings for different product components)

The detailed model provides a set of phase-sensitive effort multipliers for each cost driver attribute. These multipliers are used to determine the amount of effort required to complete each phase. The detail model also provides a three-level product hierarchy - effects that vary with each bottom level module, are treated at the module level; effects which vary less frequently, are treated at the subsystem level, and effects such as total product size, are dealt with at the system level.

Use the basic or intermediate models, as the increased effort required for the detailed model is not worth the investment for the purpose of objectively measuring improvement in the planning/estimating process.

e) Function Point Analysis:

Function Point Analysis is briefly mentioned in section 3.4.1 under the productivity metrics heading. Unlike the traditional lines-of code counts, Function Point Analysis measures the size of the problem rather than the size of the program. It measures a system's logical process characteristics, such as the number and complexity of its internal logical files and external inputs and outputs.

The calculation of Function Points is further described here. Alan Albrecht [Albrecht83] states that the function points measure is accomplished in three general steps:

- Classify and count the five user function types
- Adjust for processing complexity
- Make the function points calculation

In the first step, complexity at the individual function level is assessed. Step two assesses overall system complexity.

To count function points, firstly count the number of functions provided by the system under five function types - the number of external inputs (eg transaction types); the number of external outputs (eg report types); number of logical internal files (eg files as perceived by the user, rather than physical files), number of external interface files (files accessed by the application but not updated by it), and number of external inquiries (types of online inquiries supported). Each identified function is classified as simple, average or complex. The count of functions within each function type is classified by complexity, and multiplied by a weighting factor which represents the usefulness of the function to users.

The following table, from the International Function Point Users Group [Sprouls90] shows an example of complexity assignment for external outputs (they rate complexity as low, average and high).

	1-5 Data Element Types	**6-19 Data Element Types**	**20+ Data Element Types**
0-1 file types referenced	Low	Low	Average
2-3 file types referenced	Low	Average	High
4+ file types referenced	Average	High	High

The table below, from Albrecht's function points calculation worksheet, shows the weighting factors that each count is multiplied by, to give an unadjusted FP count by function type. These are then summed to give the total unadjusted FP count.

Type ID	Description	Simple Complexity	Average Complexity	Complex Complexity	TOTAL
IT	External Input	.....*3=.....	.....*4=.....	.....*6=.....	.....
OT	External Output	.....*4=.....	.....*5=.....	.....*7=.....	.....
FT	Logical Internal File	.....*7=.....	.....*10=.....	.....*15=.....	.....
ET	Ext Interface file	.....*5=.....	.....*7=.....	.....*10=.....	.....
QT	External Inquiry	.....*3=.....	.....*4=.....	.....*6=.....	.....
FC			Total Unadjusted Function Points:		.....

For step two, estimate the degree of influence of each of 14 characteristics, on the value of the application to the user, as listed in the table below. The degree of Influence measures are from 0, for 'not present, or no influence if present', to 5, for 'Strong influence, throughout'. Sum the 14 Degrees of Influence and calculate the Processing Complexity Adjustment factor, as follows:

PCA = 0.65 + (0.1*Total DI Points)

ID	*Characteristic*	*DI*		*ID*	*Characteristic*	*DI*
C1	Data Communications	.....		C8	Online Update	.....
C2	Distributed Functions	.....		C9	Complex Processing	.....
C3	Performance	.....		C10	Reusability	.....
C4	Heavily Used Config	.....		C11	Installation Ease	.....
C5	Transaction rate	.....		C12	Operational Ease	.....
C6	Online Data Entry	.....		C13	Multiple Sites	.....
C7	End User Efficiency	.....		C14	Facilitate Change	.....
PC					***Total Degree of Influence***	.....

To Calculate the total Function Points, multiply the unadjusted Function Points (FC) by the Processing Complexity Adjustment.

FP = FC * PCA

Albrecht based his theories on Halstead's software science formulas (see section 4.1a), demonstrating that software science formulas originally developed for small algorithms only can also be applied to large applications [Albrecht83]. He shows that function points can be interpreted to mean the weighted sum of the top level input/output items, eg screens, reports, files, that are equivalent to (n_2^*), where (n_2^*) in software science terms is the no. of conceptually unique inputs and outputs in an algorithm, and for applications, Albrecht says can be interpreted to mean the sum of overall external inputs and outputs to the program.

Function Points Analysis was developed to estimate the amount of effort required to design and develop custom application software. An early Function Points Analysis based on a project's initial requirements definition can give developers a good ball-park estimate of its size. FPs can be used instead of Lines Of Code, as a general measure of programmer productivity, as it does not suffer from the same limitations (see section 3.3.2 and section 3.4.1).

There are very few tools available to help calculate the project size in Function Points, which can mean a lot of time is spent by the organisation concerned. A study, involving Productivity Analysts in McDonnell Douglas [Bock92], reported that it frequently takes 40 hours to count a medium-sized system (800 - 2400 FPs). Another drawback is that the counting can be subjective in assigning complexities to function types, which means that two individuals performing an FP count for the same system are unlikely to generate the same result.

To overcome these difficulties, several simpler methods have been researched and presented recently. One of these methods, known as the FP-S method [Bock92], eliminates the function-type complexity classification, thus eliminating the subjectivity of the counting procedure in step one of Albrecht's method. With the subjective evaluation removed, the process can be more easily automated. The second method uses logical models, (Entity-Relationship Models and Data Flow Diagrams), as the basis of the counting process [Kemerer93]. This approach uses the E-R model to count Internal Entities as Logical Internal Files, and External Entities as Logical External Interfaces. The DFDs are used to identify External Inputs, External Outputs and Inquiries.

[Davis92] states:

The bottom line is that function point analysis is beginning to transform the black art of estimating into something more like an engineering discipline

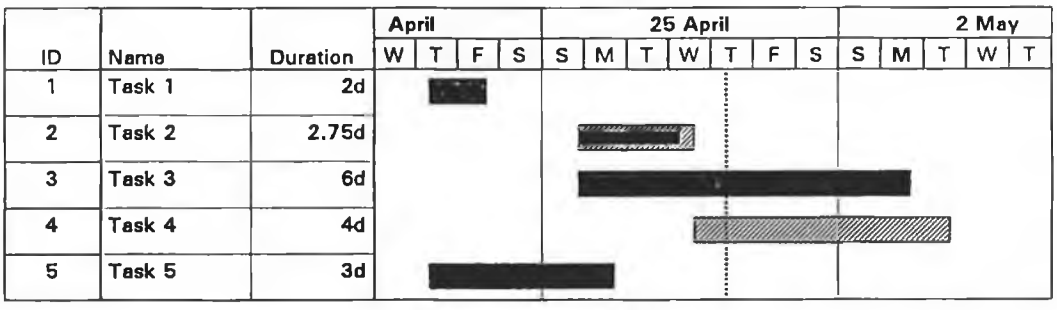
4.2 *Project Management Measures*

4.2.1 Schedule:

Cheops' Law states:

Nothing ever gets built on schedule, or within budget

At their simplest, schedule metrics show progress and slips as a measure against the original project completion/release date. This can be shown graphically by a simple Gantt chart. The Gantt chart below shows percent completion (dark line) for tasks on a project which started on 22nd April, with a dotted line showing 'today's date', 29th April. The dark bars represent non-critical tasks, whereas the bars with diagonal lines represent tasks on the critical path.



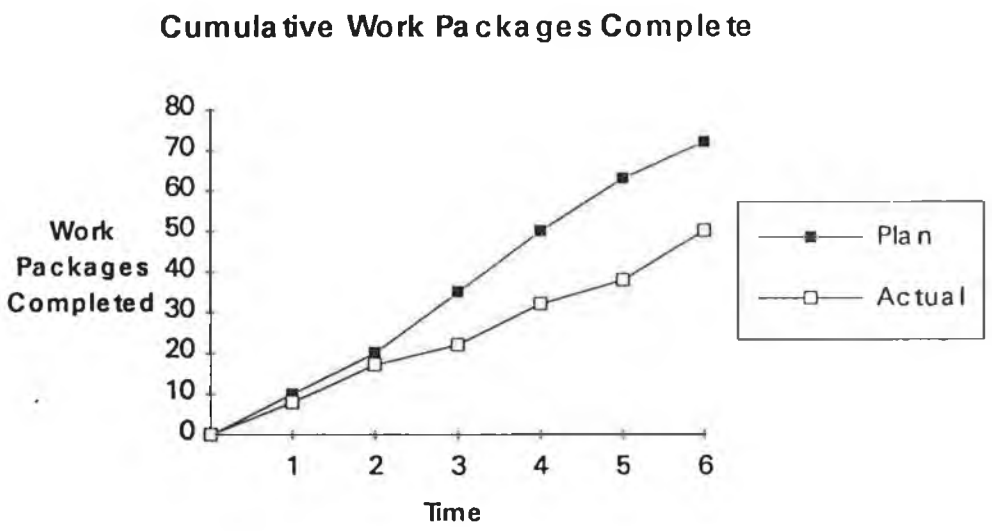
The main measure here is to calculate the percentage over-runs on the project, or at each major milestone. This is particularly useful where a company has identified generic milestones for all projects. Improvement targets can then be set against particular milestones - these improvements may be a combination of planning, productivity and rework improvements.

The two categories described below, cumulative work packages complete, and progress curves categorisation, require a lot more thought and effort on the part of the Project Manager upfront, at the start of the project.

The graphs produced by schedule metrics show per-milestone progress (y-axis) represented in percentile or the absolute value and of timeframe (x-axis), either day, week or month being selected according to management requirements.

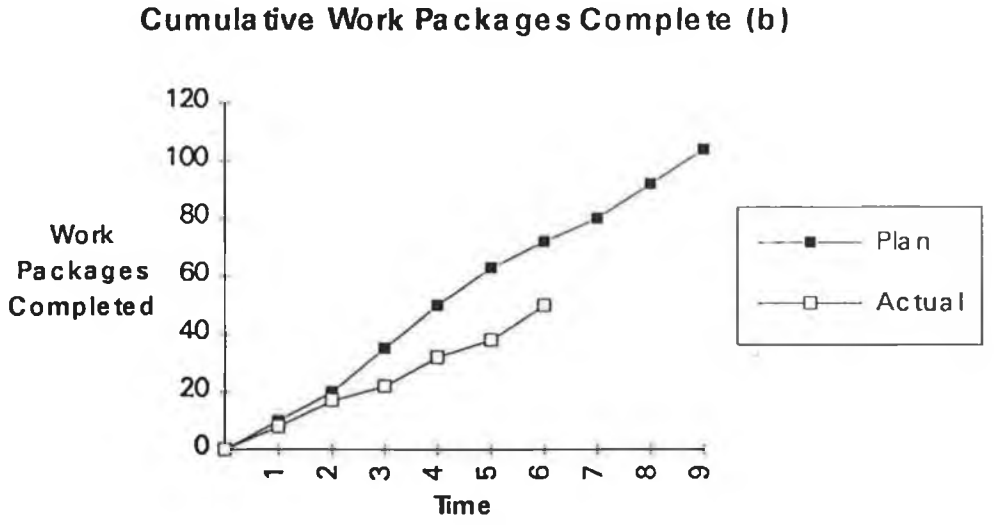
a) Cumulative Work Packages Complete

The work to be completed is broken down into sections of equal size in terms of time to complete, known as work packages. The cumulative work packages complete metric is measured by total work packages completed by a point in time. The actual versus planned total number of work packages complete is charted. [SEI91a] describes this metric in detail, and includes a section on some variants that can be used, such as measuring rework and measuring progress per development phase.



The graph above shows that from the start of measuring, the actual progress was less than the plan. However, the rate of progress has increased for period 6, which shows that the actual and the plan curves are beginning to converge, rather than diverge further. The primary use of this graph is to determine whether or not the planned completion date for the project is realistic. At any point in time, the percentage variation of actual vs. the plan can be obtained. As the process improves, the variation between actual and plan should decrease, as the process comes under full statistical control.

By charting the plan for the full duration of the project, and the actual work packages completed to date, and extrapolating the slope of this curve out, the likelihood of meeting the scheduled completion date can be deduced (see chart below). The example used shows that the schedule will not quite be met, at current progress rates. The options are to either change the planned completion date by one and a half time periods, or else increase the number of work packages completed by 50% (to 18 per time period), by changing staffing levels, experience levels, etc.



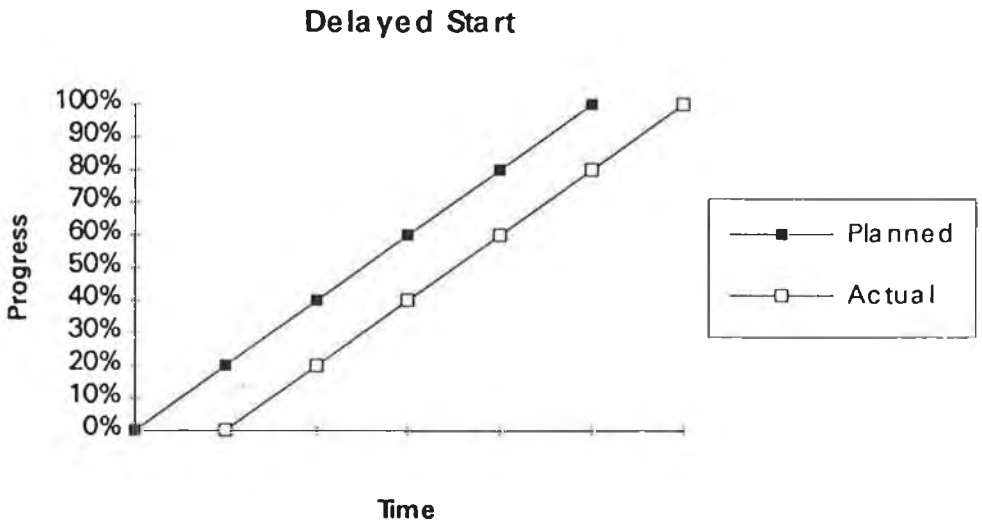
b) Progress Curves Categorisation

NEC Telecom Systems, Japan [Kadota92] analysed various types of discrepancy between plans and actual progress curves, and found that the characteristics of progress curves, despite their great variety in appearance, could be classified into either one of six patterns, or a combination of them.

The six patterns are delayed start; progress delay increase; progress plateau due to work interruption; progress drop due to rework; progress stagnancy due to work difficulty, and slow progress at earlier stages. Each identified pattern also has suggested counter measures, so that when a pattern is identified as the project progresses, the correct countermeasure can be taken. This is the optimum implementation of continuous process improvement.

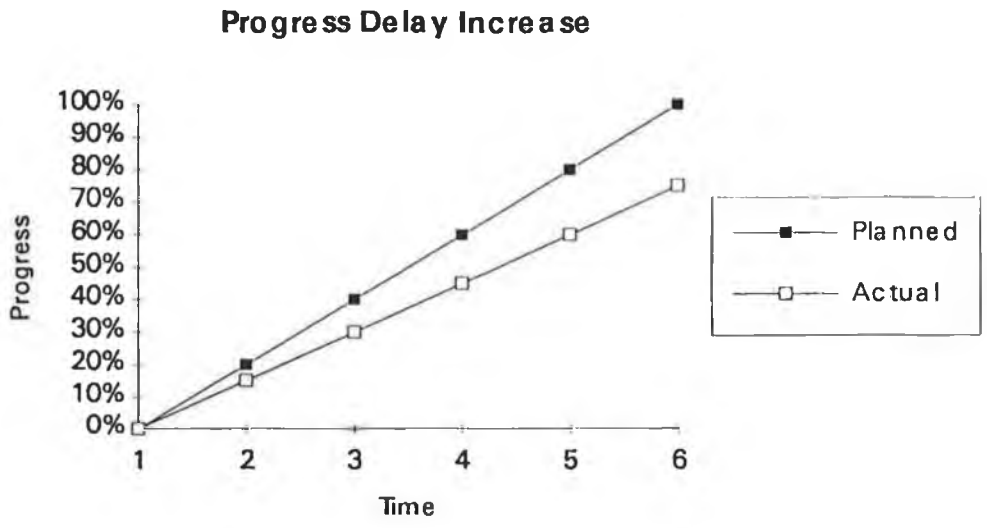
The completed graph, after project completion, can be used in the project post-mortem analysis, to identify learning points, and to establish areas where improvements can be made for the next project. Of utmost importance is to effectively plan the project before it commences, based on previous experience and from the information the metrics provide.

Pattern A: Delayed start



This situation results from delayed receipt of deliverables, resource availability, etc. In other words, work from previous stages is not yet complete.
The countermeasures are to clarify completion of the previous phases and expected hand-off dates, and to negotiate earlier with those responsible for the previous phase to urge deadlines to be met.

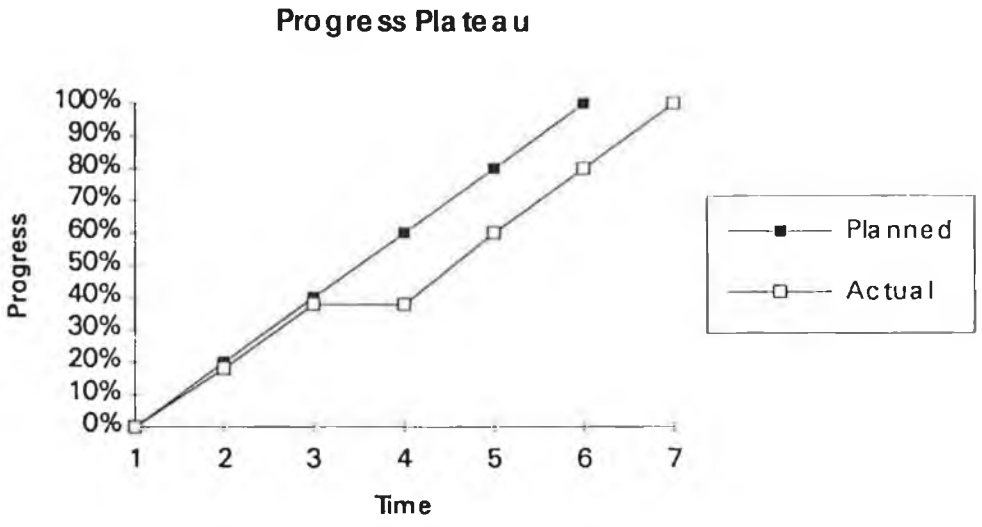
Pattern B: Progress delay increase



The discrepancy between the plan and the actual graph increases, which is a result of mismatching effort and productivity. It occurs when people are less experienced than expected, or when there are many unforeseen complex issues involved in the project.

The countermeasures are to increase effort (eg. overtime) when the complex issues have delayed productivity, or assign experienced people where the cause is lack of experience.

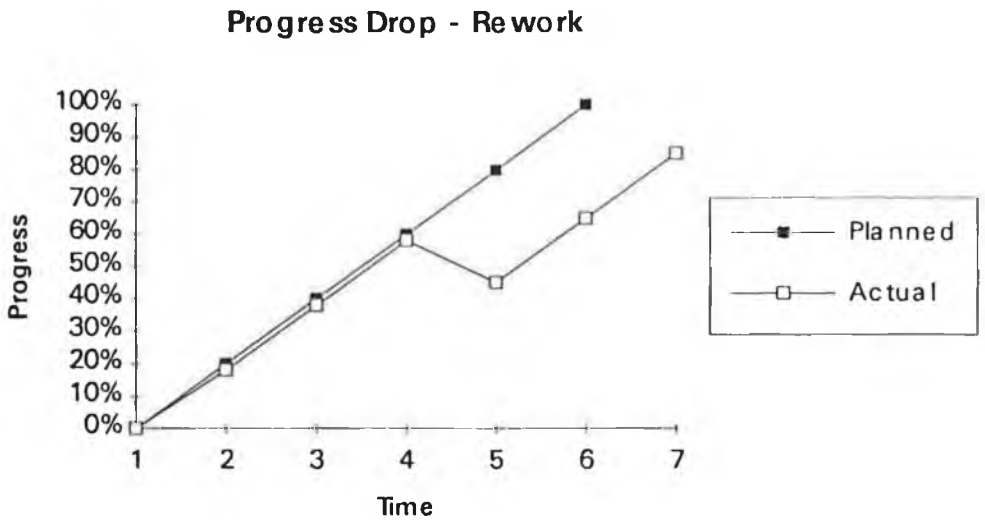
Pattern C: Progress plateau due to work interruption



This occurs when people are assigned to higher priority projects, urgent support work, and their manager also assumes they are working on the current project. Everyone on the project will be working to their full capacity, yet the project is not progressing.

The countermeasures are to watch everyone's daily work contents carefully, monitor interruptive work, reassess priorities of all tasks and monitor these tasks/priorities on a weekly basis, and revise people's work assignments. If effort is being tracked, watch out for the support activities, or the time spent in non-project specific areas.

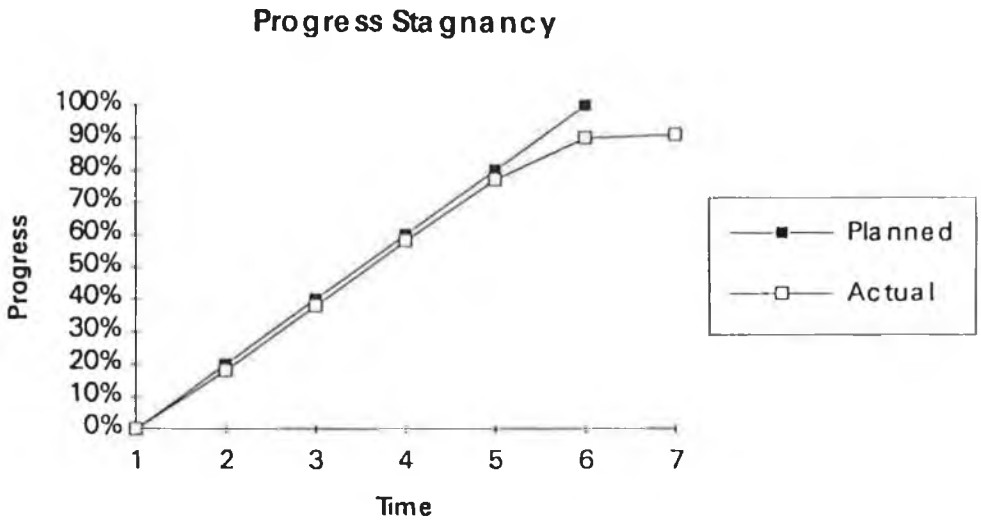
Pattern D: Progress drop due to rework



This pattern is shown when additional work occurs which originates from either a previous phase or the current phase. Progress drop can occur when a careless Developer hands off modules to testing which are regarded as 'finished', but without first unit testing them. This occurs when a tight milestone deadline approaches, and rather than miss the deadline, the Developer skips testing, and delivers 'on time'. The modules, previously marked as finished, may be returned to the Developer for proper completion.

The countermeasures here are to examine methods and contents of the reviews in the previous stage, if this is the cause, or to introduce frequent reviews in the current phase, if this is the cause. Everyone should check the quality of their work thoroughly before it goes to the next phase.

Pattern E: Progress stagnancy due to work difficulty

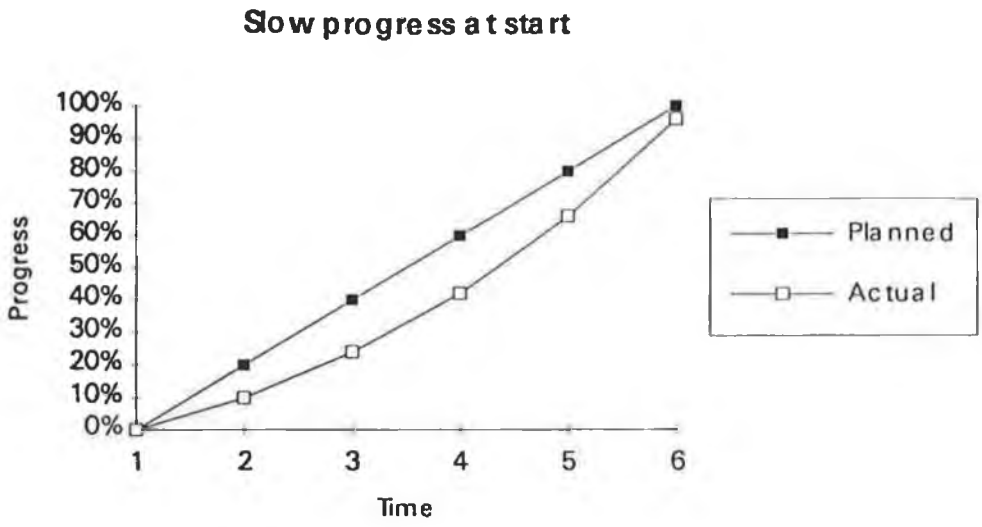


This is the classic 90% complete forever chart. The progress curve is consistent with the plan until the project is 90% complete, followed by increasingly slower progress. Major causes are complex technology which is unfamiliar to the project personnel, and left unresolved; and work starting with the necessary conditions unspecified. The ninety-ninety rule of project schedules is appropriate here:

The first ninety percent of the task takes ninety percent of the time,
and the last ten percent takes the other ninety percent.

The countermeasures are for experienced and skilled people to join or support the project team, and also to confirm conditions, ie plan effectively, at the start of the project.

Pattern F: Slow progress at earlier stages



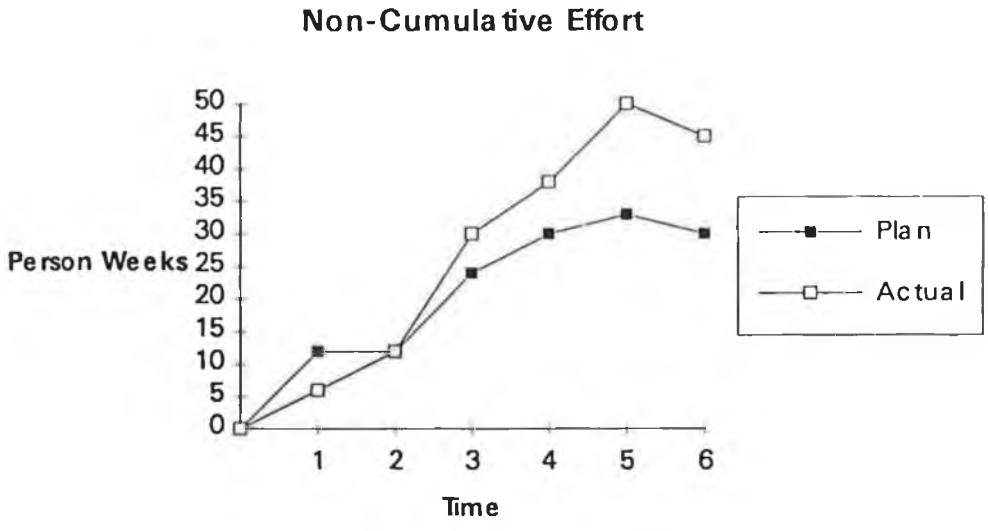
Insufficient understanding at the start of the project causes progress delay, and as time goes by, the progress speed increases, as the project team gains familiarity, and as undefined conditions settle.

The countermeasures are to specify undefined work contents as early as possible, and ensure that adequate training and support are given to all members of the project team.

4.2.2 Effort:

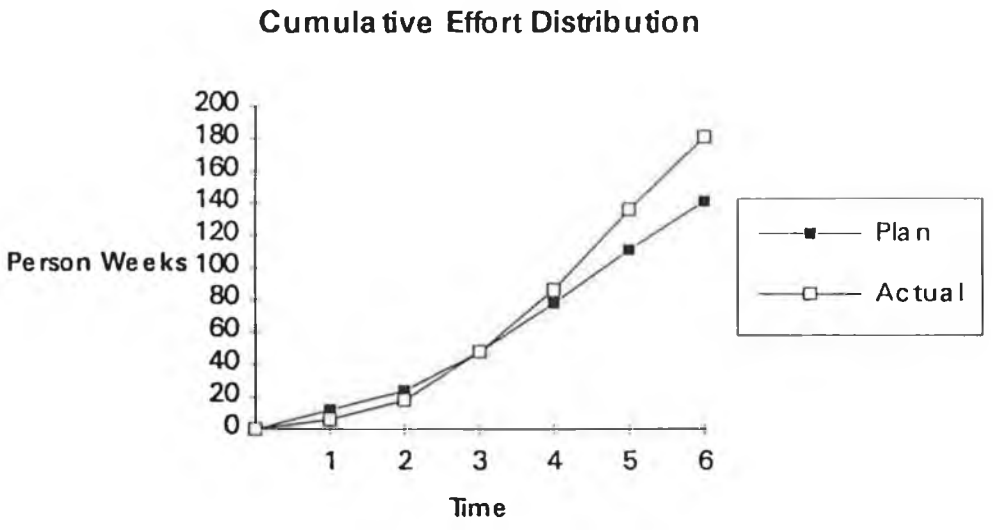
Effort metrics show the man-hours per time period (normally per month). Two variants described in [SEI91a] are Non-Cumulative Effort Distribution, and Cumulative Effort Distribution.

Non-Cumulative Effort Distribution shows the effort per time period, so that peaks and troughs can be seen. This can also be used for cost accounting purposes by multiplying the effort for each job function by the manmonth cost (including overhead costs), which gives the project cost for each month.



The graph above shows that the project was understaffed at first, then became overstaffed in order to try to catch up on planned progress. Comparing this graph with the Cumulative Work Packages Complete graph in section 4.2.1 (a), gives a good picture of the current and future project performance.

Cumulative Effort distribution enables the relationship between total actual effort expended and total planned effort expended at a point in time, to be viewed. This metric is used to determine a project's performance towards meeting the planned amount of effort.



The graph above shows that at time period 3, the amount of planned effort equalled the amount of actual effort for the project. Referring back to the schedule progress graphs in section 4.2.1 (a) again, it is seen that only approximately two-thirds of the work packages planned were actually completed in time period 3. This says that although the planned amount of effort has been expended, the progress is only about 66% of planned.

Comparing the Cumulative Effort Distribution (CED) metric with the Cumulative Work Packages Complete (CWPC) metric identifies the following four key conditions explained in [SEI91a]:

a. CED actual > CED planned and CWPC actual > CWPC planned
b. CED actual <= CED planned and CWPC actual > CWPC planned
c. CED actual > CED planned and CWPC actual <=CWPC planned
d. CED actual <= CED planned and CWPC actual <= CWPC planned

Condition a is quite optimistic, in that schedule is better than planned, with effort being greater than planned. The expectancy here is that the project would be completed ahead of schedule, with the expected total amount of effort planned (which would show actual CED higher than planned for each time period).

Condition b is the best condition - the project completed ahead of schedule with less effort than planned.

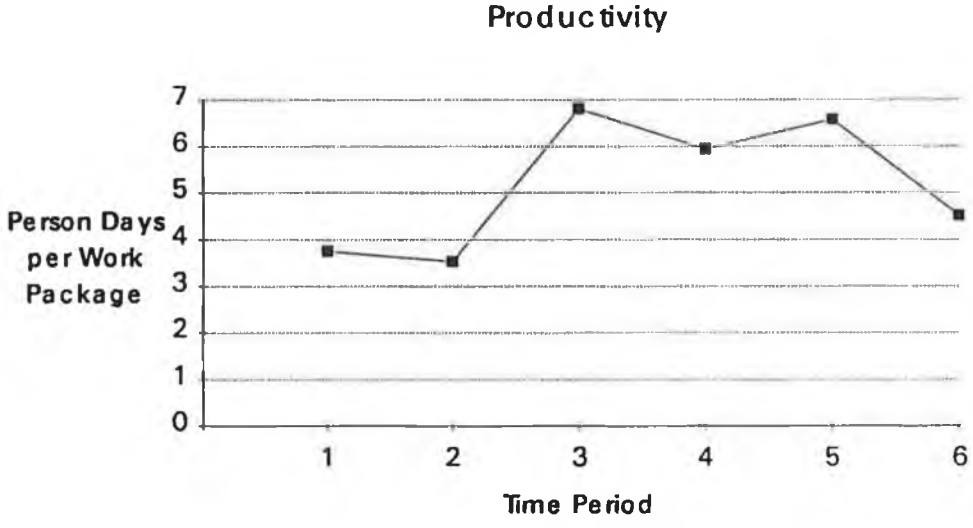
Condition c is probably the most frequent condition observed - less schedule progress than expected, with more effort than planned. This leads to delayed project completion, and extra staff on the project or excessive overtime.

Condition d shows less progress than expected, but also indicates an understaffing problem. The planned completion date will not be met, yet the project may well be completed within the planned amount of effort.

4.2.3 Productivity Measures

These measures show how the underlying process can affect the progress of a project. Process improvement efforts can be demonstrated by the use of productivity metrics. However, care must be taken when determining productivity measures to use (section 3.3.2 explains why Lines of Code are not a useful measure of programmer productivity). Another potential pitfall is the fear of individuals being measured. Productivity measures should be a measure of the total effort expended against the total progress to date for each major activity.

Productivity measures can be in the form of the number of days per amount of work (eg 4 days per work package), or the amount of work completed per person per time period (eg 2 work packages per person week). Function Points can be used as the basis for determining work packages, as described in section 4.1(e). The example below uses the Non-Cumulative Effort Distribution divided by the Work Packages completed per time period, expressed in person days. For productivity metrics of this type, the work packages must be roughly equal in size w.r.t. effort required for completion.



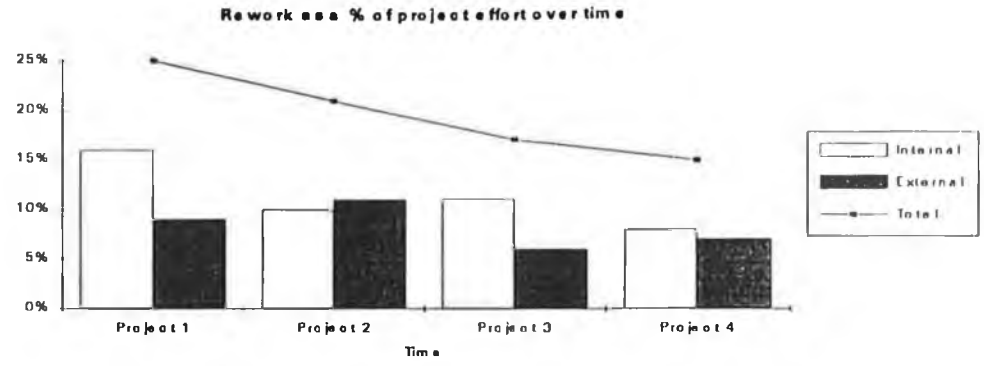
4.2.4 Rework:

In a software development environment, rework is expressed in terms of time (or cost) to amend systems due to requirements changes, or in order to correct errors found. It is one measure that often goes unnoticed and unmanaged within the development process, possibly because rework is expected - it has become an integral part of the process. The uses of rework measurement are described briefly in section 3.4.1 (c). Mosher's Law of Software Engineering states:

Don't worry if it doesn't work right. If everything did, you'd be out of a job.

Rework metrics are tied in very closely to schedule and productivity metrics. Unexpected changes in requirements cause rework which will affect both schedule and productivity measures. (see chart of progress due to rework, Section 4.2.1(b) - chart D). Errors in project specifications are a major cause of rework, identified during the testing phase. Each rework cause needs to be addressed, and the cost of rework should be measured in conjunction with the schedule, productivity and quality measures.

Probably the best way to measure rework is to measure time and effort spent per rework cause. The basic measures recommended are a total measure of rework per project, expressed as a percentage of total project effort, and as a percentage of total project time. The next level of granularity suggested is to split the time and effort spent on rework into its major constituent causes. The most basic implementation would be to categorise rework cause into internal (ie due to errors introduced by the project team itself) and external causes (due to customer requirements changes, etc.). Working on eliminating internal causes is easier, since the team has direct control over these. If the team takes a right-first-time approach to specification at the requirements definition stage, the 60% of defects reported that originate in this stage, such as missing, erroneous, ambiguous or conflicting specifications, can be eliminated [Oates92].



5. QUALITY MEASURES

As explained above, software defects make a significant direct contribution to the total rework costs of a project. Measurement of the defects can help the team to understand where and how they occur, and by pinpointing the problems, can provide guidance in detecting, preventing and predicting defects. Defect measurement is one of the most important metrics for process improvement, as it provides a direct measurement of the process and products.

The terms Quality and defects are both difficult to define. The SEI have defined a software defect as any flaw or imperfection in a software work product or software process [SEI92-TR22]. The terms *software work product* and *software process* are defined in [SEI91-TR25].

> A *software work product* is any artifact created as part of the software process, including computer programs, plans, procedures and associated documentation and data.
>
> A *software process* is a set of activities, methods, practices and transformations that people use to develop and maintain software work products.

The number and frequency of problems and defects associated with a software product are inversely proportional to the quality of the software [SEI92-TR22].

Carole Jones [Jones85], says that internal test problems are very expensive if you consider the cost of screening the problem, debugging it, developing and applying the fix, and verifying its correctness. She says this normally costs a minimum of nine hours for the most simple problem, when all factors are considered.

5.1 Defect reporting

In order to manage defect measurement efficiently and effectively, the defects should be written to a database in a structured manner. Several texts suggest formats for bug reports - [Jones85] describes a format for IBM; [SEI92-TR19] describes in detail a method for the DoD, which includes checklists for clearly defining defects. The description provided here is based on the bug-reporting mechanism in Microsoft.

Each defect goes through a life-cycle of Active - Resolved - Closed. A defect is active when it is first reported, and remains active while it is being evaluated. When it has been fully evaluated, and fixed or otherwise dealt with, it is resolved, and assigned back to the Tester who Activated it for rechecking, and the Tester then closes the bug. Each field in the database is described in accordance with each of the defect lifecycle phases:

ACTIVE

Bug Number:	The system assigns a bug number, in sequence, to each bug entered
Status:	Assigned to Active, Resolved or Closed, according to defect lifecycle phase
Opened date:	The date the bug was entered
Opened by:	The name of the person who opened the bug
Title:	A unique title for the bug
Environment:	The full hardware and OS configuration used
Severity:	Select severity from 1 to 4. Severity 1 indicates a crash/data loss; Severity 2 indicates impaired functionality of the product; Severity 3 indicates cosmetic problems, and severity 4 indicates a trivial bug such as minor cosmetic problems.
Build Number:	Type in the software build number the defect is reported against
Priority:	Urgency to fix, rated from 1 to 4, where 1 is top priority, ie essential, and priority 4 is 'fix if time'. The priority of a bug depends on the development phase and the proximity of the release date. A priority 4 close to release may be a priority 3 earlier in the process.
Description:	A full description of the defect is provided with detailed steps to reproduce it included

Defects are categorised according to 6 keywords, which are named and defined by the Test Manager before the database is set up. Each keyword has a set of standard values, from which one value is selected when entering the defect report. The six keywords were chosen specifically to enable the defect metrics information to be obtained easily through a series of standard reports. Four of the keywords (area, sub-area, type and test case) are used when opening a defect report. The other two (origin and time) are filled in at the resolution phase.

Area:	The highest-level product component, eg. .exe, helpfile, install
Sub-Area:	Describes the sub-area of the selected component in which the bug was found.
Type:	Describes the bug type, which gives an indication of the bug cause. Examples of values for type would be text appearance; mathematical function; discrepancy between program and specification; crash/hang; error messages, etc.
Test Case:	The number of the test case used to reproduce the bug

RESOLVED

Resolution:

Fixed. The problem has been fixed, and will not reoccur. The Fixed Rev field must also be filled in, and a full description of the fix given.

Duplicate. The bug has already been reported. The Related Bug field must also be filled in.

By Design. The software works according to its design. This is not a bug.

Not Repro. The bug cannot be reproduced. More information needed.

Won't Fix. The problem will not be corrected. It may be either too difficult to fix with the available resources or too trivial to worry about.

Postponed. The problem will be resolved in a later product revision.

Fixed Rev:	The build number of the software containing the fix
Related bug:	The number of the other bug of which this bug is a duplicate
Resolved date:	The date the bug was resolved
Resolved by:	The name of the person who resolved the bug
Origin:	Stage of development where the error was created
Time:	Time taken to evaluate and resolve the defect

CLOSED

Closed date:	The date the bug was closed
Closed by:	The name of the person who closed the bug

5.2 Defect Metrics

Keeping track of the defects found is the most important element of defect metrics. Consistency of reporting methods between projects helps to compare metrics across several projects. This section concentrates on the use of metrics to predict bug densities, which in turn helps to determine when to stop testing.

a) Halstead's bug prediction formula

A lot of work has been done to predict defect density from the program size and/or complexity. In section 5.2 (a) several of Halstead's quantities are described, which are based on the numbers of operators and operands in a program. One other formula he devised is the bug prediction formula:

$$B = (N_1 + N_2) \log_2 (n_1 + n_2)/3000$$

where:

- n_1 = the number of unique operators in the program (eg. keywords)
- n_2 = the number of unique operands in the program (eg. data base objects)
- N_1 = the total occurrences of operators in the program
- N_2 = the total occurrences of operands in the program

Boris Beizer, [Beizer84], uses an example of a program which accesses 75 database items a total of 1300 times, and which uses 150 operators a total of 1200 times. The expected number of bugs for this program would be:

$$B = (1300+1200) \log_2 (75+150)/3000$$
$$= 6.5 \text{ bugs}$$

Beizer says that there is solid confirmation of the correlation of the predicted bug count using Halstead's metric, and the actual bug count. This equation provides a good rule-of-thumb measurement, but for determining when to stop testing, it's better to use Musa's software reliability models, which are described further on in this section.

b) McCabe's Complexity Metric

McCabe's complexity metric is described in detail in Section 5.2 (b), for ascertaining the complexity of a module. This section illustrates how this cyclomatic complexity number can be used to aid testing.

The cyclomatic complexity of a module can be expressed as a number, or drawn as a flow graph. A cyclomatic complexity of greater than 10 can mean a defect-prone module. Modules that have a cyclomatic complexity of less than 10 are well-constructed with the resultant expectancy of a low defect density.

In [Ward89], McCabe describes how the cyclomatic number and the accompanying program control flow graph can be used to identify test cases for the well-known triangle graph problem, where three integers are entered, and the system determines what type of triangle they represent. He says that the cyclomatic complexity number corresponds to the number of test paths, and these in turn correspond to the basic paths derived from the control flow graph. From this information, the test cases for the program can be generated. For example, a program with a complexity of six would mean there are six test paths for the program. Each test path is then identified, by reference to the control flow graph. Finally, the test conditions for these test paths are written, from which the test data can be generated to satisfy these conditions.

Ward, [Ward89], claims that the test-case generation capability of the McCabe methodology has been very useful, in the Waltham Division of Hewlett Packard, in establishing rigorous module testing procedures. He says the cyclomatic complexity values have been used as an indicator of which modules should be subjected to the most testing by the test group. The most extensive testing is performed on modules with abnormally high complexity values.

c) Musa's Software Reliability Measurement

Software reliability measurement is a statistical process to provide quantitative guidance on the reliability of a system with respect to execution time. John Musa uses execution time as the basic dimension of reliability measurement because it accurately reflects software stress:

> *A piece of software that is never executed never fails*
>
> [Musa89].

Software failure occurrence is modelled by a Poisson process - a Poisson process can be characterised by its expected value function. In software reliability measurement, this is the cumulative number of failures expected to occur by the time the software has experienced a certain amount of execution time.

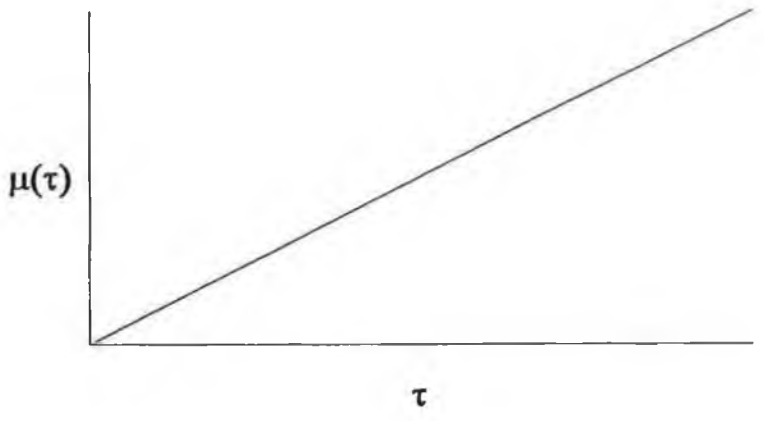
Musa describes three models, depending on whether or not the faults found are fixed, and the impact that fixing the faults has on the software. The first model (*static execution-time model*) is for software that does not change, eg firmware. In this model, the likelihood of failure as execution time increases is constant, since the same bug can reoccur. The second model (*basic execution-time model*) is for software where faults are being corrected when they are found, and assumes that all faults are equally likely to cause failures. The third model (*logarithmic Poisson execution-time model*) is similar to the second in that faults are corrected, but assumes that some faults are more likely to cause failures than others, and by fixing these, there is an exponential improvement in the number of failures w.r.t. execution time.

The variables used in the functions are τ, which represents execution time, and $\mu(\tau)$ which represents the cumulative number of failures. The failure intensity function, which is a measure of the instantaneous rate of failure, is denoted by $\lambda(\tau)$.

Musa's models help determine when to stop testing. The method is quite simple to follow - select test cases; record, at least approximately, the amount of execution time between failures, and continue until the required failure-intensity level has been met to the desired level of confidence. The quality of the software can be quantified in statistical terms, ie. the probability of 1,000 CPU hours of failure-free operation in a probabilistic environment can be stated.

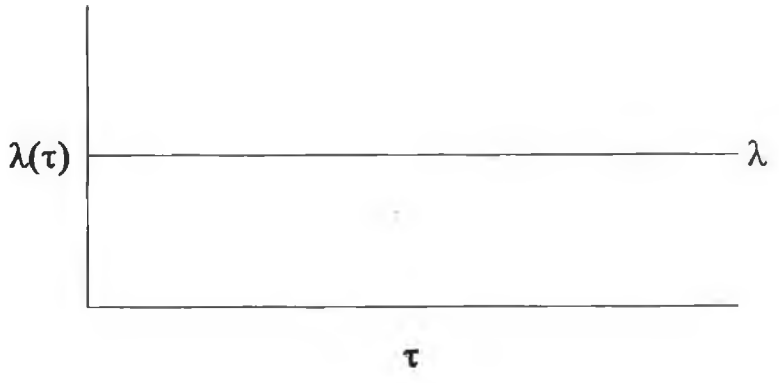
In this model, the software is not changing as defects are found. Therefore, the number of defects increases linearly with time.

$$\mu(\tau) = \lambda\tau, \text{ where } \lambda \text{ is a constant.}$$



and, since the software is unchanged after defects are reported, the likelihood of a failure occurring remains constant.

$$\lambda(\tau) = \lambda$$



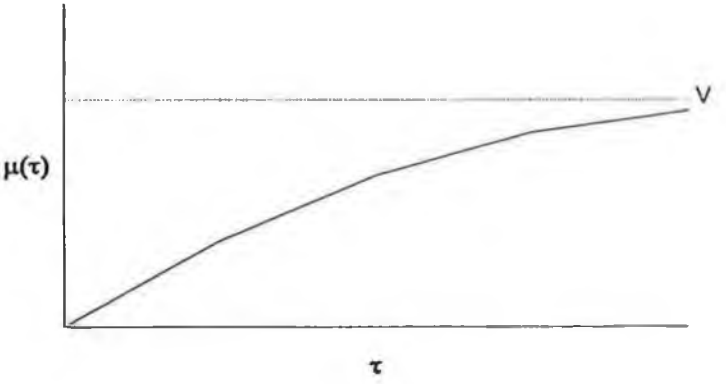
In the second model, the faults in the software are being fixed, and are equally likely to cause failures, which means that failure intensity should decrease by the same amount whenever a correction is made. The function is:

$$\mu(\tau) = v_0 [1-\exp\{-(\lambda_0/v_0)(\tau)\}]$$

where:

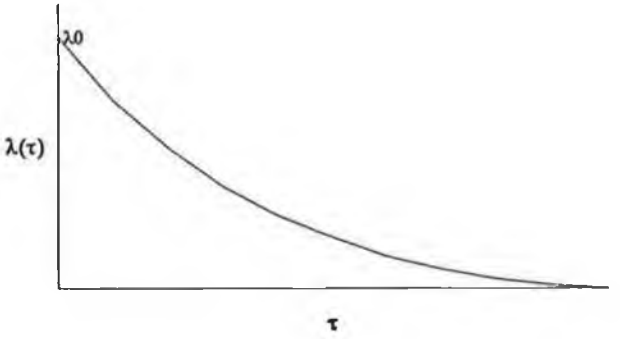
λ_0 is the initial software failure intensity at the beginning of the observation period

v_0 is the total number of failures that will be experienced in an infinite amount of execution time



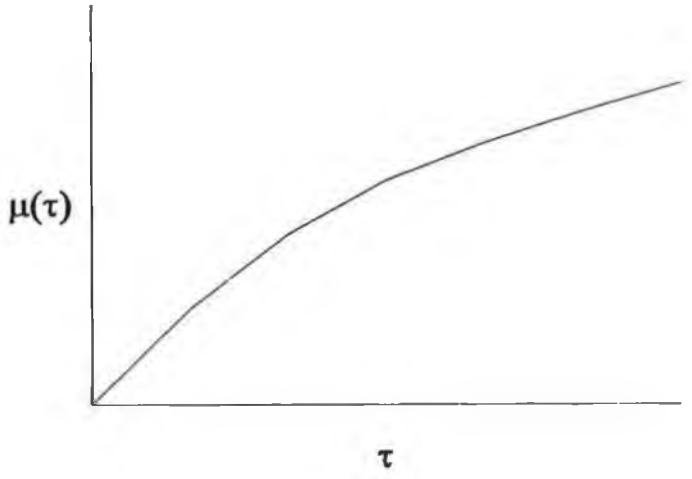
and for failure intensity versus execution time:

$$\lambda(\tau) = \lambda_0 \exp [-(\lambda_0/v_0)(\tau)]$$



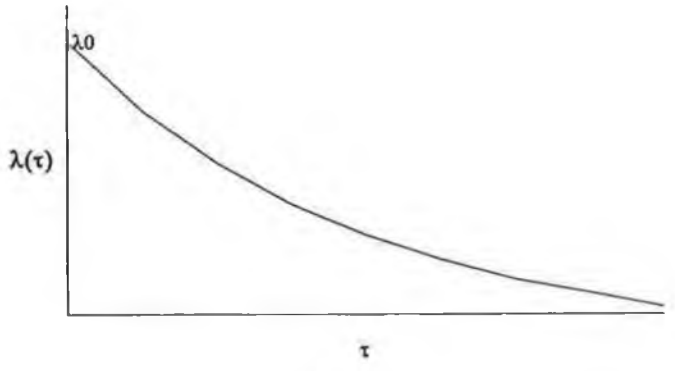
For this model, where the improvement in failure intensity with each correction declines exponentially as corrections are made,

$$\mu(\tau) = (1/\theta) \ln (\lambda_0 \theta \tau + 1)$$



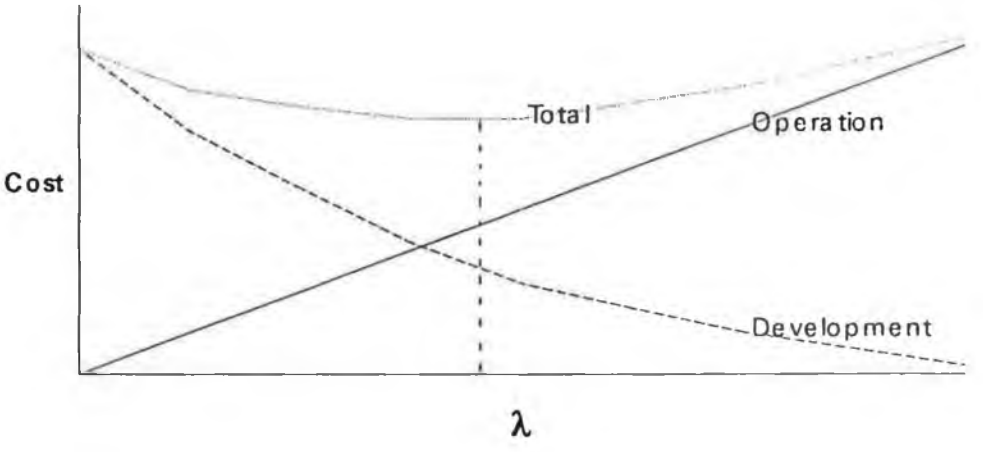
and failure intensity is defined as:

$$\lambda(\tau) = \lambda_0 / (\lambda_0 \theta \tau + 1)$$



The required failure-intensity objective should be the optimal figure that minimises the overall life-cycle cost of failure. A very low failure-intensity level (high quality) will have high software development costs, whereas one that is too high (low quality) will have high maintenance costs. To select a failure-intensity objective, select the lowest total cost, and drop a vertical line to the x-axis, where the line crosses the x-axis is the optimum failure intensity level.

Selecting a failure-intensity objective

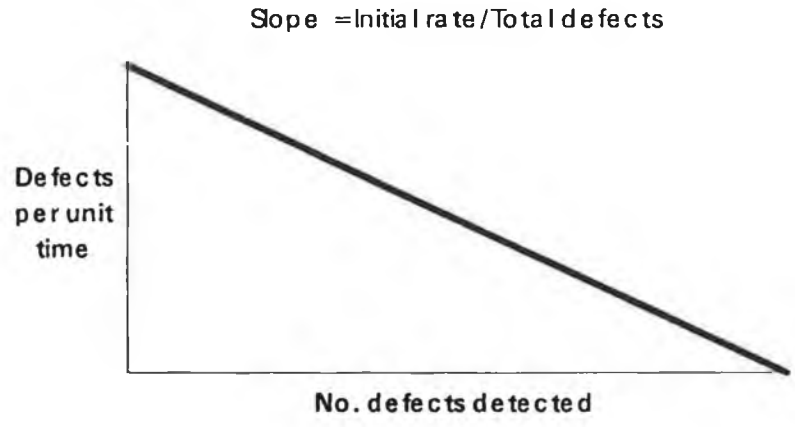


d) Using defect rates to predict product stability

Knowing when the product is stable can be a difficult task. James Walsh, [Walsh93] says that one of the great mysteries of any software development project is how many bugs are left in the program - the number of bugs found to date on a project is simple to obtain from bug reporting records, but estimating the number of bugs remaining in the product is a much more difficult number to quantify.

So how can defect densities be used to determine when the product is ready to ship? One method could be to divide the number of defects fixed by the expected number of defects in the module. An answer near to one indicates a stable system.

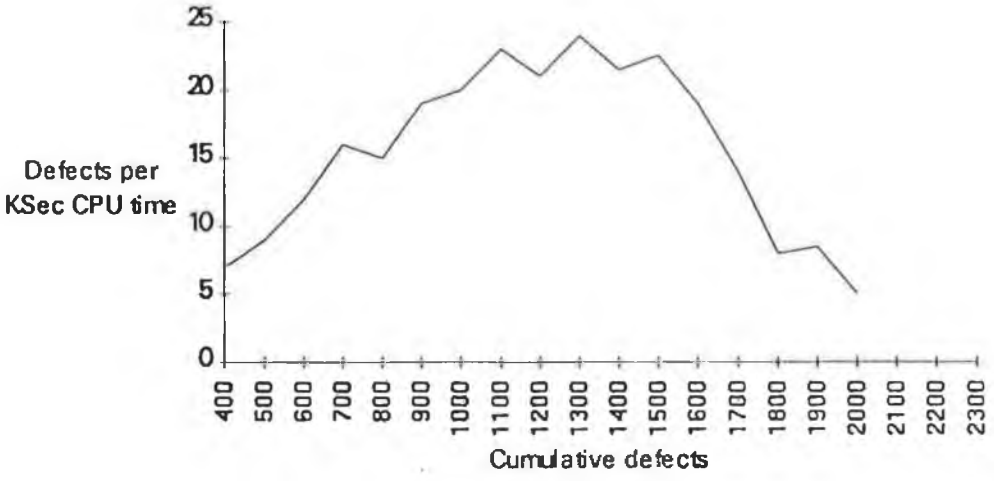
A more precise method involves using Musa's basic model, but graphing defects per unit time against the total number of defects found. From this, the total number of defects in the program can be predicted, and therefore the number of defects remaining to be found. This model suggests that if a straight line is drawn through the curve describing the decline in defect rate, this line will intercept the x-axis at a point corresponding to the total number of defects in the program (ie, the slope of the line = Initial rate/Total defects). In the chart below, the initial defect discovery rate is on the left-hand-side, and the total number of defects occurs where the line crosses the X-axis.



James Walsh, [Walsh93], uses Musa's model to measure the defect discovery rate (number of new defects found divided by the amount of testing time) and its rate of change, in order to predict the total number of bugs in a project he worked on, known as the Rational Rose project. He found that there are three patterns within the defect discovery rate curve, corresponding to three product phases - integration, alpha test, and beta test. The curve has a leading edge, where the discovery rate is rising, a plateau where the discovery rate is constant, and trailing edge where the discovery rate rapidly declines.

The defect discovery rate is initially low, as there are likely to be several serious crashing bugs present during the integration phase, preventing some of the underlying system to be tested until these bugs are fixed. The plateau during alpha testing is due to the rate at which the bugs can be found and reported by the test team - the breadth of the plateau, rather than its height, gives an indication of how buggy the software is. The sharp decline in the defect discovery rate during the beta test/customer ship phase is due to the extra testing required to find the fewer remaining bugs in the product.

Defect discovery rate as a function of cumulative defects

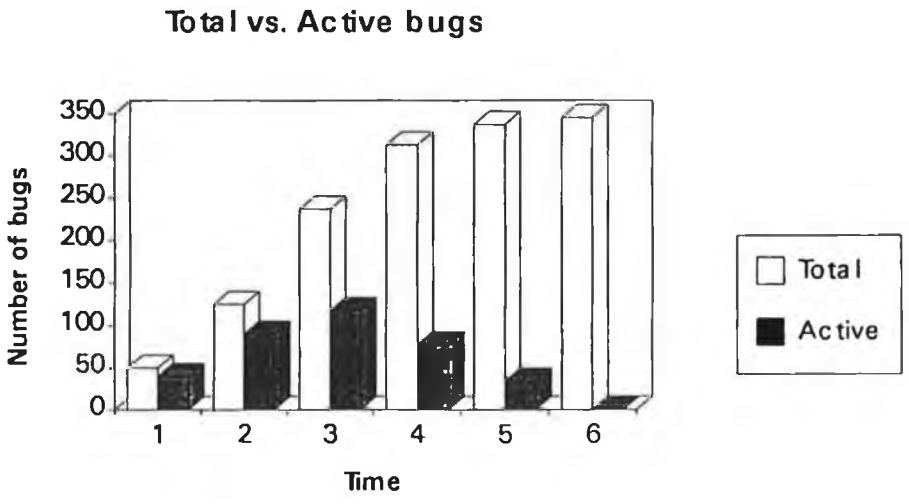


By graphing the defect discovery rates over the project life-cylce, against the cumulative number of defects discovered, the number of defects remaining can be inferred.

e) Distribution of Active defects over time

The software reliability metrics above are very useful for determining the number of bugs in the product and when to stop testing. However, these metrics take some effort to implement, and execution time, which is the basis for the metrics, can be difficult and cumbersome to measure. A quick and efficient way to start measuring defects to help determine the stability of the product is to measure active defects against the cumulative defects reported. The length of time a defect remains active is also a good indicator of quality (and of developer productivity). This is also known as the 'age' of the defects. As the release date approaches, the number of new bugs found should show a decreasing trend, whilst the number of Active, ie open, bug reports should diminish towards zero (ie the 'total bugs' trend upwards should slow down dramatically, and the 'active bugs' trend downwards should increase).

In a process improvement environment, the number of bugs reported in the next version should decrease, and the number of active bugs should always remain low, with a shorter average 'age' of bug.

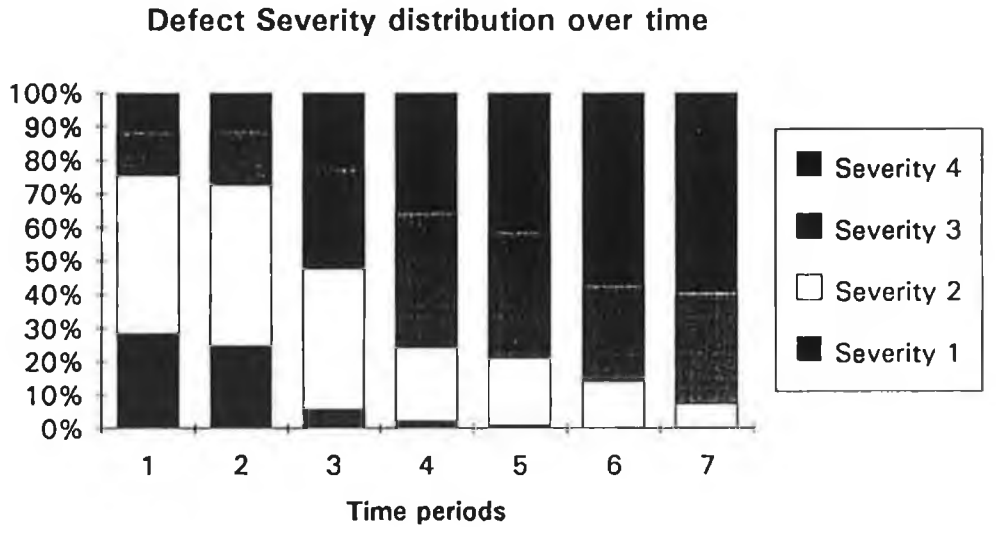


f) Defect Severity

Another simple measure is to measure defect severity over time. A severity 1 defect involves a system crash or serious data loss. Severity 2 represents major loss of functionality. Severity 3 defects are minor functionality problems. Severity 4 defects represent trivial errors or cosmetic defects.

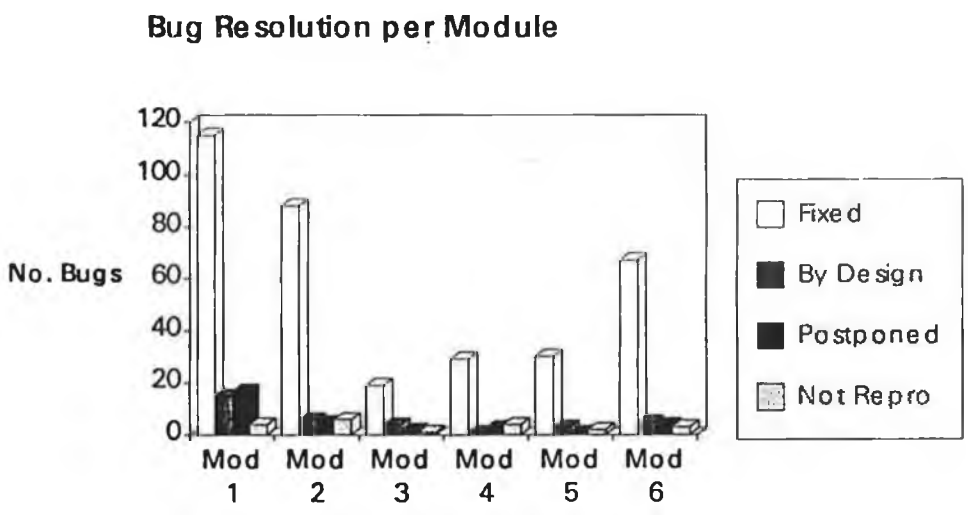
As the release date approaches, the severity distribution should move from predominantly severity 1 and 2 bugs reported, to predominantly severity 3 and 4 bugs. Also, from one version of a product to the next, the number of severity 1 and 2 bugs should decrease, assuming one of the process improvement goals is to say, decrease the number of severity 1 and 2 bugs found by 20%.

The following chart shows the relative percentage of each bug severity at each time period throughout the test phase. This chart could also be displayed in absolute numbers, rather than as a percentage.



5.3 Defect Resolutions

The following chart shows bug resolution per module. The interpretation of the categories is: Fixed means that the program was altered to fix the defect. By design refers to the fact that this is not a defect, but a feature of the program - it's supposed to work this way. Postponed refers to those defect that cannot be fixed in this release, but will be fixed in a maintenance release. These defects are usually severity 4 (trivial) defects, such as an untidy looking message, which is much too expensive to fix close to the release date. Not repro refers to a defect that was not reproducible by the developer to whom it was assigned, in the same build number as that for which it was reported.

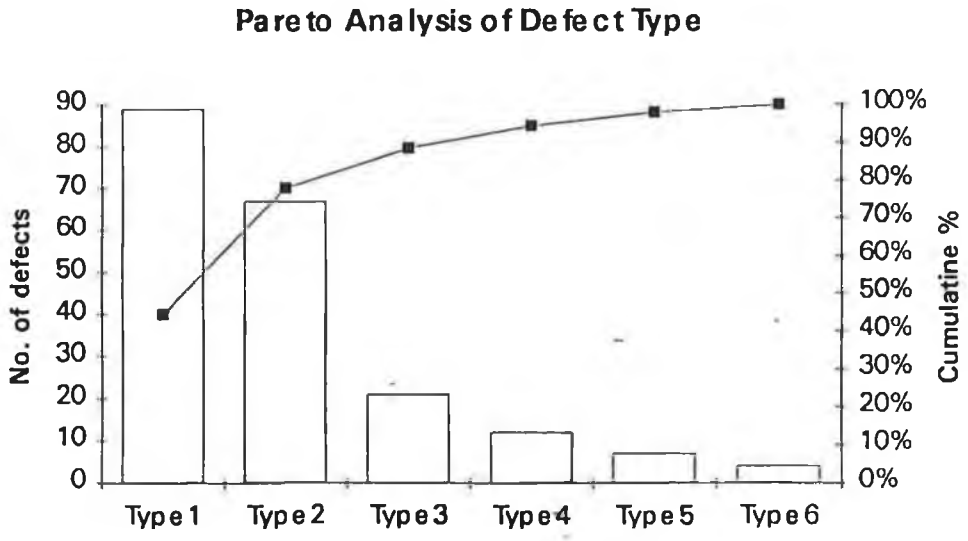


5.4 Pareto analysis

Pareto analysis is a method of organising data to highlight the major factors that make up the subject being analysed. It is based on the 80-20 rule; 80 percent of the problems result from 20 percent of the causes. It can be used for a variety of defect metrics - where I find it most beneficial, is in analysis of defect type. This helps to highlight the 'vital few' areas that must be addressed, as opposed to the 'trivial many'. Pareto analysis of bug type is used to eliminate the most common bug type. An alternative chart could be to do a pareto analysis of bug cause (ie why the bug was introduced), then to work to eliminate that cause for the next version of the product.

To construct a pareto chart:

- Identify defect types to be used
- Categorise the defect data into the selected defect types
- Place the types in decreasing order of magnitude
- Calculate the percentage of the total defects for each defect type,
- and the cumulative percentages (starting with the highest percentage)
- Draw the bar graph, so that:
    - The left y-axis represents the count of actual data
    - The right y-axis represents the percentage of total defects
    - The bars represent the number of defects
    - The line represents the cumulative percentage

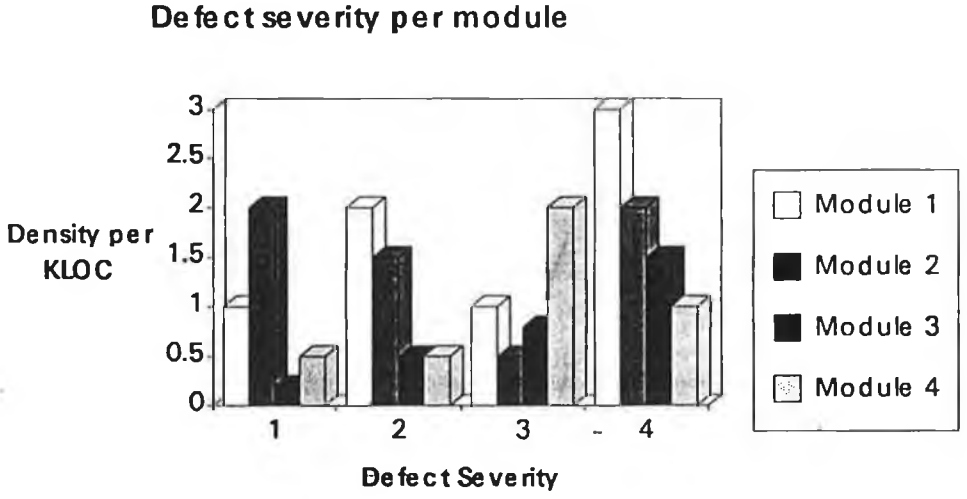


5.5 Measures/Metrics after project completion

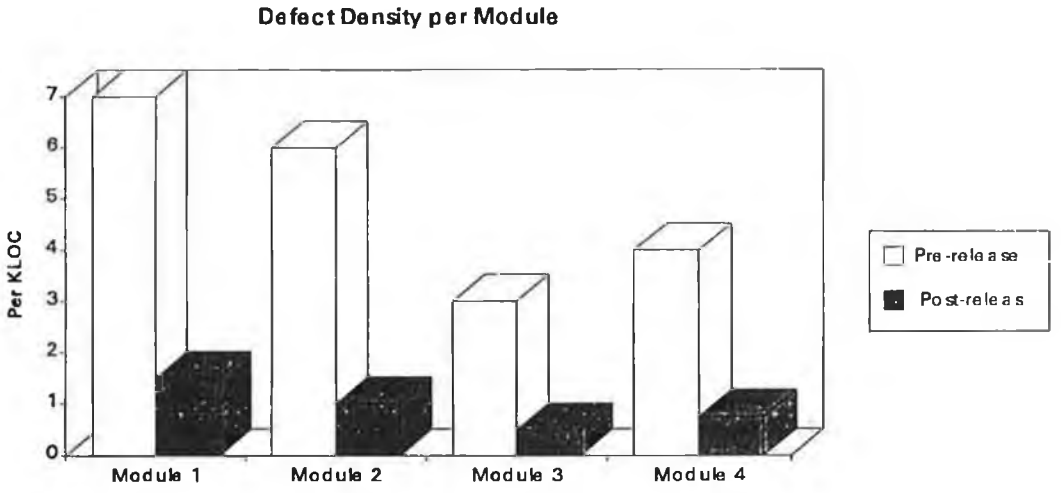
The project post-mortem is where a transfer of learning takes place. It is where the problems and issues of the project are highlighted, so that they can be eliminated, helping to ensure the same problems and errors do not re-occur in the next project. End-of project metrics use the same categories as described for in-progress measures. Quality metrics are probably the most important, and easiest to obtain, measures for the end-of-project analysis. The highest-level post-release quality metric is the number of re-releases and maintenance releases during the productive lifetime of the product.

If someone is watching their weight, they are advised to weigh themselves at the same time of day each time, and preferably no more often than once a week. Similarly, it is easiest to compare projects using post mortem data, in order for objective comparisons across projects and levels of improvement against the previous project to be accurately assessed. The project post mortem provides the real view of process improvement, and also provides the data as a baseline from which further improvements will be sought.

Rather than having time as the x-axis, post-project measures are a static representation of the project in its entirety, and will normally have the values of the measurement category itself, or each module, as the X-axis. The following graph shows the relative severity of the bugs found in each of four modules of a product.



The defect count does not diminish to zero once the product has been released. It goes into another phase of testing, with the customers reporting bugs as they use the system. Post- release Defect Density (no. bugs per KLOC), shows the number of defects found by the customer(s) after release, and can be shown on a graph, with pre-release defect density, as shown below:



5.6 Causal Analysis

Many companies use the defect metrics above to determine error-prone modules or process stages. IBM have gone a step further, as described in [Jones85]. They take full advantage of their data by augmenting these metrics with an in-depth study of the errors themselves and what causes them. Once they have evaluated the root cause of each error, they put plans in place to remove these causes. This process is referred to as Causal Analysis, which is centred on three concepts:

- Programmers should evaluate their own errors
- Causal analysis should be part of the process
- Feedback should be part of the process

To enable implementation of the above three concepts, the project team should hold causal analysis sessions at the exit step of each process stage. An action team should also be set up, to ensure management and implementation of the suggested improvements. The format of a causal analysis session, using the defect data from the defect database, would be as follows:

1. **Compare defect results** against acceptable quality standards.
2. **Evaluate all defects** - i.e. read through, discuss the defect, its category and the cause, ensuring all team members understand why the error occurred and what the root cause was. Error causes are likely to fall into the following broad categories:
   - *Communications* (breakdown of communications within the team, from the customer, across functions, etc.)
   - *Education* (can be further subdivided into misunderstanding of a function, lack of tools knowledge/training, misunderstanding of the development process, lack of specific technical knowledge)
   - *Oversight* (where everything is not considered, eg. an error condition is missed)
   - *Transcription* (where the Programmer knows and understands fully what to do, but for some reason just makes a mistake, eg types in the wrong label)
3. **Create an action list** of error prevention actions to be taken, by whom and the date for completion. This list is generated by asking how could the error have been avoided? and what corrective actions are recommended?
4. For common errors, **create a 'common error list'**, to be used at each project start-up, and which can be accessed by project teams throughout each project.

6. CASE STUDY: IMPLEMENTING METRICS

The above twelve steps were undertaken in my case study - a Software Localisation environment of just over 400 employees, in which PC software packages are 'localised' into up to 13 different languages. Localisation involves adapting the software and documentation for a particular local market, and includes the translation of the user interface, changing national language support settings (date, time, currency formats, etc.) and adapting examples to fit in with the different cultures. eg. a blueberry muffin company would be a great example company to use for a US-bound product, but would need to be changed to, say, an example of a pasta company for Italy, and a car manufacturer for Germany.

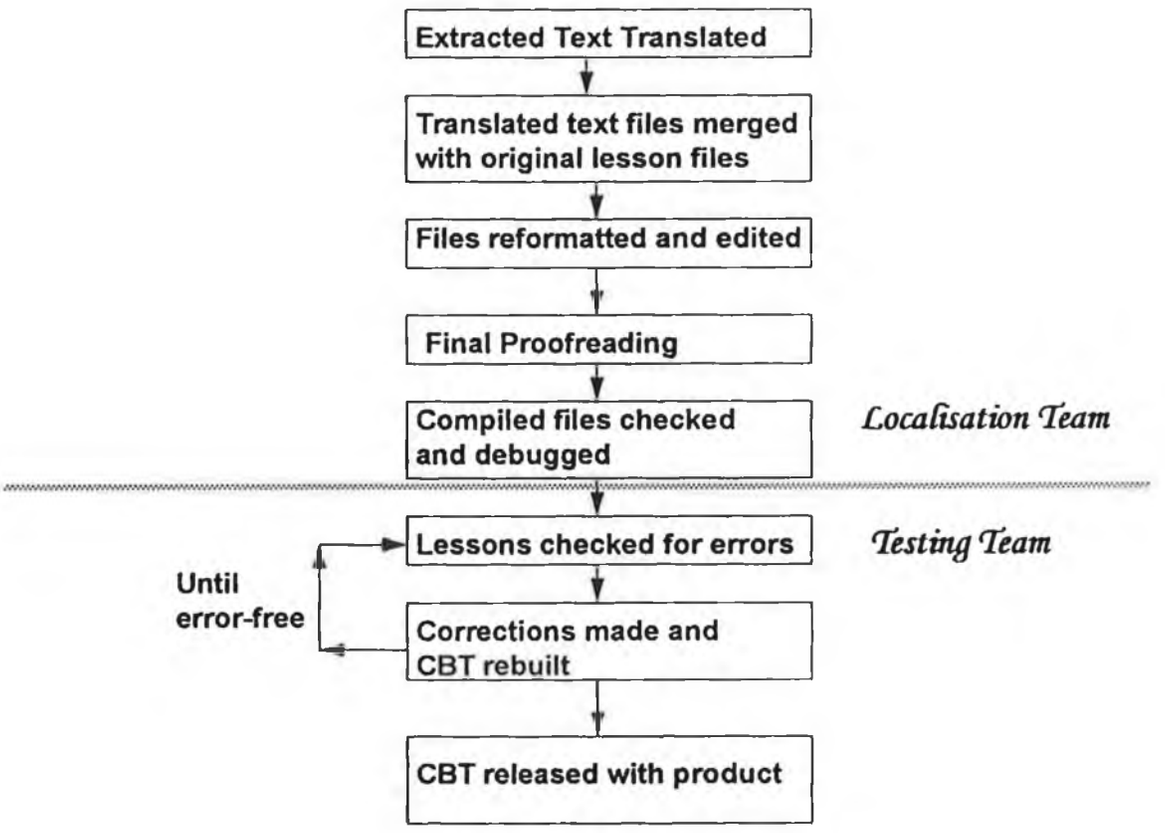
A cross-departmental metrics team was set up to introduce a set of measures that would firstly determine the current status of the localisation process, and which could subsequently be used to support process improvement efforts. The set of measures would be used to identify areas for improvement, and would then be used to demonstrate and quantify improvement for the next product version. The metrics themselves only provide information, and process improvement efforts are a different entity, which should have a positive effect on the measures that are recorded. A comprehensive set of measures covering all aspects of the localisation process was required, so that a full picture from planning to release could be obtained, including the costs, and to provide the ability to identify when things start to go wrong so that counter-measures can be applied as early as possible. At the time the team was set up, the aim was to get measuring all major activities as soon as possible, with a time span of six months seen as the life of the metrics implementation project. The metrics decided on were presented within the time period allotted, however because of the size of the company, it took an additional six months to get agreement from team leaders and Managers that this is what they wanted to measure, to make some amendments to the measures, and to implement measurement as one of the weekly/monthly activities.

The metrics that were chosen for the localisation process are fairly similar to those that apply to a straight-forward software development process, although several cannot be used. Since localisation is performed after much of the code has been written, and does not involve changing the code, the specification and code measures described in section 4.1 are not applicable to a localisation environment. This chapter reports on each of the 12 steps explaining how each step went, and how the metrics were used, in the case study.

6.1 - Map the software development process

Several versions of a process map already existed, from the efforts of the Software Process Improvement team, the Documentation Localisation team, and the Methods Group in the US parent company. Therefore, the Metrics team did not have to draw a process flowchart from scratch. Post-It notes were stuck to the wall, which also helped eliminate some of the steps that had been identified in previous process maps. The team kept to a fairly high-level view of the process. Looking at the Capability Maturity Model, the company was identified at level 2, although a few projects were at levels 1 or 3.

Because of confidentiality issues, the full localisation flowchart cannot be reproduced here. However, a high level view of the localisation of a Computer Based Training (CBT) module is shown:



Of importance to the Metrics team were the individual processes within the overall localisation process, and identification of customer/supplier interactions where files and information are exchanged (so-called hand-offs). Measurement at hand-off stages helps in schedule measurement and quality measurement.

At the very highest level, the processes identified were the software process and the documentation process. The Computer Based Training and Help modules are considered as on-line documentation, and follow a hybrid process between documentation and software. They are in here as following the software process, although they are normally regarded as following the documentation process with a testing phase added. The main reason for including online documentation with the software process here is that the Quality metrics described for software are equally applicable to the help and Computer Based Training modules.

Software	*Description*
Planning/preparation	involves preparing localisation and test plans; setting up the localisation compile kit; extracting files and strings to be localised, preparing schedules.
Localisation	of the main program; the helpfiles; the Computer Based Training program; the Setup program, and any other addins.
Testing	of all components that have been localised and recompiled.
Bug Fixing	of all bugs found during testing.
Releasing	of disks to manufacturing for mass duplication.

The major handoffs identified were from the US to Ireland at the planning/setup phase, from Localisation to Testing and from Testing to Manufacturing.

Documentation	*Description*
Planning/preparation	vendor selection; preparing documentation for translation; preparing schedules.
Translation	of the documentation by the vendors
Formatting	of the translated documentation
Art Preparation	taking screendumps of the localised software, and placing into the translated documentation
Review	of the documentation - translation, formatting and artwork
Release	of film to the printing Vendor via Manufacturing.

The major handoffs were from the US to Ireland, from the Translation Vendor to User Education, and from User Education to Printing.

6.2: Define the Corporate Improvement Goal

The long-term qualitative goal defined in [Metkit92a] was adopted, which also states that this goal should be the goal of any company, no matter what their business is, which makes it a very general goal, needing further definition:

> Continuous improvement of all processes which leads to
> better use of our resources, improved efficiency of our processes,
> improved productivity of our project teams and
> improved quality of our products.

In further discussions, it was decided that the primary goal was to produce localised software products that meet the users' expectations, and that do not contain any bugs that were introduced as a result of the localisation process. It was also discussed that the users' perception of the product's value depends on when it is available to them (ie how soon after the US version of the product has released). The four main quality goals are to consistently produce localised products which:

- meet customer requirements
- do not contain additional bugs that impact the users
- are localised within reasonable cost constraints
- are localised efficiently and timely

The only way to ensure continued user satisfaction is to continually improve the process so that the company produces localised products of the required quality (zero localisation bugs which impact functionality in any way), within the shortest timeframe after the US version has released, and at a reasonable cost. In order to improve the areas of the process where 20% of the improvement effort will produce 80% of the gains, specific measures must be introduced throughout the process.

The ultimate aim is to increase the number of localised versions that can be released within two months of the release date of the English product, by streamlining the process in terms of quality, time and costs. There are several goals which quantify the number of products that should be released within 30 days and within 60 days of the English language product, with specific timeframes specified in each of these goals.

6.3: Conduct an Employee and a Customer survey

These surveys were conducted to get a general impression from both the employees and the sales subsidiaries of their perceptions of the current practices. The survey also asked respondents how important they think each of the elements are to the future success of the company. The employee surveys were collated and put in a spreadsheet, from where a summary was produced in order of priority (as described in section 3.3.2) and an analysis of what the results mean was written for each statement. A difference of more than 1.0 between current practices and importance shows that these areas need to be worked on. The data was collated for each job function, and for each Department (Product Unit) as a whole. The following shows an example of the type of information collated and reported for a Product Unit.

1. Resource planning Diff: 1.4
2. Metrics Diff: 1.25
3. Realistic Schedules Diff: 1.2
4. Defect prevention Diff: 1.1

5. Project Post Mortems Diff: 0.95
6. Standard methods/procedures Diff: 0.9
7. Quality planning Diff: 0.9

8. Timely process training Diff: 0.8
9. Project planning Diff: 0.8

10. Schedules are updated regularly Diff: 0.75
11. Subsidiary Evaluations Diff: 0.7
12. Timely tools training Diff: 0.7
13. Subsidiary involvement Diff: 0.65

14. Risk Assessment Diff: 0.4
15. Standard tools Diff: 0.3

Survey results such as these demonstrate the need for improvement in resource planning and scheduling, acknowledges that metrics really need to be implemented, and shows that defect prevention is an area that requires some work. Therefore,

with results like these, metrics efforts should be focussed on quality (ie defect metrics), effort, and scheduling.

The 'customer' survey is a different type of survey, in that it is ongoing and not a once-off survey. It gives a measure of the quality of each localised product. The survey consists of a product section, and a process section, and is filled out by each of the subsidiaries about three months after the relevant product is released.

There are two levels of rating - a top level, which gives a rating of 1 to 5 per product area, and a detailed level, giving a rating of 1(strongly disagree) to 4 (strongly agree) against a list of statements. The top-level table is reproduced here. Refer to Appendix B for a sample of the full subsidiary evaluation form..

Please rate the quality of the product below by area as follows:

5 = EXCELLENT
4 = GOOD
3 = SATISFACTORY
2 = IMPROVEMENT NEEDED
1 = UNACCEPTABLE

Evaluation per Product Area	Quality Rating
Software	
Packaging	
Help	
Printed Documentation	
CBT	
Timeliness	

Both the employee and subsidiary surveys ask the respondents for their subjective opinions. However, a collection of these subjective opinions still gives an objective measure. By repeating the surveys in six months' or a year's time, the improvement, if any, in each of these areas will be quantified. The internal project metrics system should also provide a meaningful objective measure of the main areas covered in each of the surveys.

6.4: Define applicable metrics categories

A) SIZE METRICS:

Since the traditional Lines Of Code measures are not applicable in a localisation environment, an alternative measure was sought that would allow us to normalise quality indicators and to compute productivity measures, across several projects, for comparison purposes.

Traditionally, there had been attempts at calculating project complexity for planning and scheduling purposes. The complexity factor was a rating from 1 to five, where 1 was easy to localise, and 5 was very difficult to localise. This rating was derived by a combination of the size of the product, the technical difficulty to localise it, whether it was a new product or an update, its known localisability problems, and which languages it was to be localised into (eg Eastern European languages had a higher complexity due to the different character set used). These complexity numbers were too difficult to calculate, still fairly subjective, and generally only useful as a guideline for strategic planning.

The measure first suggested for size was the number of translatable strings in a product. Some strings have only one word which is easy to translate, whilst others are a full line of text, which take longer, but it was reckoned that overall it would work out fairly evenly across projects. After some further thought, it was decided that the number of translated words might be a better estimate of size to start with. Both words and strings were used for the first metrics, and then evaluated to see which one gives more accurate estimations. At the time of deciding to use these measures, there was no quick way to count words or strings, so a simple tool was written that counts both words and strings to be localised from a directory of files. This was a good starting point. A complexity rating could be introduced at a later stage to take into account the number of dialog boxes, menus, hotkeys, etc. which have to be localised as well as the words, ie to count the number of words, and add to this, the number of menus and dialogs which have been multiplied by a weighting factor, as localising menus and dialog boxes involves more effort than just translating words. In mathematical notion, this would be:

$$Size = W + Mx + Dy$$

Where: W= Words; M= Menus; D=Dialog Boxes;
x, y = weighting factors

For help files, the measure of size was words of translation, which can also be converted into approximate numbers of pages and screens. Complexity also comes into play with help files - the number of bitmaps to be localised adds to the complexity. With the help, measures chosen were similar to those for the software - start with words and pages as the unit of measure, and perhaps move on to using some form of complexity weighting factor at a later phase in the improvement process.

For Computer Based Training (CBT) modules, ie tutorials, the number of screens was used as the size to start with, but this changed to number of words, to fit in with the way cost was defined. It is difficult to compare CBTs across projects, as some are much more detailed and more technically complex to localise than others, and they often use entirely different sets of tools. Without a sensible complexity rating system, comparison of some CBT measures across projects are difficult, and straight comparisons of effort and productivity across different departments cannot be made.

The aim in defining size measures is to use them as the basis for all estimations and calculations throughout the process. Thus, the cost estimates, defect density calculations and productivity rates will all be based on the same sizing information. Each category of use, eg pricing, headcount, scheduling, productivity etc. used to report figures against a slightly different unit of size, and a full cohesive picture was difficult to get from each of the pieces of information. As an example, help size could be referred to in words, pages, screens, or KBytes. The aim was to change this, and to encourage standardisation on one size measure for all categories.

B) PRODUCTIVITY

When planning, managers tended to subconsciously apply productivity measures to estimates. For example, they'd say 'it takes one manweek to format 250 pages of documentation, therefore this product has 2000 pages, so we'll allow two man-months'. The testing effort estimation was less scientific, although it always worked out close to what had been planned - here, they'd say 'it's a big project, which we haven't localised before, being done in 6 languages, so going from past experience, let's have a team of 8 people for 8 months'. For the purpose of the metrics, this was formalised - if process improvements are introduced, then it is necessary to be able to state the benefits in terms of increased productivity, within a reasonable timeframe.

As stated in section 3.4.1 (b), productivity metrics on their own are too simplified - if one can format 250 pages of documentation per manweek, then why not put 8 people on 2,000 pages of documentation for one week? In the testing example, if 6 languages of a large project can be tested and released in 8 months with 8 people, then why not do it in 2 months with 32 people, assuming non-concurrent testing of the languages? The simple answer for the testing example is that the testing & bug-fixing cycle cannot be compressed by so much, ie to about 1.5 weeks per language. Also the communication overheads with this number of people would be quite ridiculous.

Care had to be taken to ensure that everyone involved realised that it was the project, not the individual, that was being measured. If people think they are about to be measured, and that these measures will be publicised, there can be some resistance. Because of the nature of the work, which is in localisation teams, people were informed that the information would be collected and collated for the team as a whole before being made available to others outside the team. The other point that was stressed was that the measures were for use primarily by the team itself. Section 7.1.3 b) contains details of the tracking system most of the departments use to track effort, from which productivity rates can be calculated.

C) REWORK

There are several causes of rework, due to the number of interactions with other groups outside of the project team's control, as well as rework due to internal mistakes. Rework on Documentation was the area that traditionally caused problems - rework could involve anything from 50% to 100% of the number of pages originally planned. A lot of rework arises out of updates from the US groups, much of which is grammatical or cosmetic changes to the documentation, and it is very time-consuming to sort out functional changes from cosmetic changes, so often all changes will be implemented in each localised version. One change in the original US version could mean at least nine (sometimes more) repeats of the change as it is implemented in each of the current localised versions.

Rework on the software side is more difficult to define - should bug-fixing time be included as rework? Can the software rework be measured in the same way as the documentation rework (ie count the number of strings rather than pages?). In a simultaneous-ship environment (ie when the US version and several other versions are shipped to customers within a few weeks of eachother), updates can be expected right up to two weeks before release.

It was decided that rework was one of the most important factors for the process improvement efforts, therefore both quantity of rework and effort spent on rework should be measured, as a percentage of the total quantity of work and project effort. From a cost perspective, the invoiced cost of rework from the translation vendors should also be recorded and reported, to give a full perspective of the rework on each project.

D) EFFORT & SCHEDULE

It was necessary to measure the effort spent on each project, the duration of the project, and at any time, know the current status of each major task/activity. This was achieved through a combination of effort and schedule measures. In order to improve the process, where the time is being spent on each part of each project, and where the greatest over-runs are w.r.t. the proposed schedule should be known. It was a general feeling that in the month before release, things become chaotic, with people generally working 60 to 80 hours each week. In order to control then improve the situation, both schedule and effort throughout the project should be measured, and then improvement targets set.

Paul's second law is an appropriate quote:

The sooner you fall behind, the more time you will have to catch up

The effort metrics would also be the basis for keeping track of the internal costs of the project, and would be linked with manmonths costs for this purpose.

The schedule metrics would tie in with the information required by the project managers to manage the product schedules.

To measure effort, it was decided to introduce a time-tracking system, where each person would enter his/her time, in days, against a list of activities/tasks. This would be rolled up each month to get the total time worked on each area of the project.

Schedule tracking was based on a project-management system as a starting point. The work breakdown structure was based on the current localisation process, and standard milestones were set for each project. The Program Managers could then track the actual schedule versus the planned schedule, and the percentage completion of each task. See section 7.1.2 c) for further detail.

E) QUALITY

The quality metrics chosen were at two levels.

The highest level are the release metrics - ie how many products were released? What percentage of products met their published delta? What percentage of products were re-released due to a process error? How many disks were released for duplication, and how many were re-released? These quantities ideally should all be zero.

The other level is the bug recording and analysis of bug type distribution. The following questions needed to be answered - what areas had the most bugs? What was the distribution of each category of bug? Did the bug-fixes require code changes or changes to the localised strings? What was the ratio of bugs found to time spent testing?

The measures implemented include defect density, defect discovery rate, bug type analysis, and bug cause analysis. Quality measures are the most important measures for identifying areas that require improvement. The benefits of new tools or methods for localisation are discussed in terms of impact on the product quality. For example, is a certain type of bug eliminated by using this tool/method? is the number of bugs reduced? is less testing time required as a result?

Section 7.1.3 b) explains each of the Quality Metrics that have been adapted for and adopted in the localisation environment, and explains their usefulness in evaluating the quality of products, and in identifying areas for improvement.

6.5: Break corporate goal into a specific goal for each category

The tangible, measurable overall goal is to be able to consistently release 20 products, with zero localisation errors, within 60 days of the English version, within a 3-year timeframe. Each sub-goal must keep this overall goal in mind.

There are a set of sub-goals in each metrics category defined, some of which have been further defined as a result of the 'measure current process and products' step (section 6.9). A general outline of some improvement goals is given here, rather than a full description of each, for confidentiality reasons:

> To reduce the amount of testing required for each project by 50% within a year, by automation and risk assessment methods, in combination with improved localisation methods and tools.
>
> To reduce the time spent bug-fixing, by dramatically reducing the number of errors introduced during the localisation process
>
> To reduce hotkey errors to zero on windows projects from the next version onwards

The above are examples of goals on which some progress has been made, since the achievements can now be expressed in terms of tangible numbers.

6.6: Define specific measures

Chapter 7 defines each measure that was implemented for each process stage. This is the most time-consuming and difficult area of the entire metrics implementation process. A quote from Albert Einstein is appropriate here:

> *"Not everything that counts can be counted, and not everything that can be counted counts"*

A variety of different measures have been tried, and as it is an evolutionary process, a company will not get it completely right at the first attempt. The aim is to take a set to start with, and then further amend this set to suit the organisational needs. The set to start with should cover the areas that require most urgent improvement. The principles of continuous process improvement can also be applied to the implementation of a metrics system.

6.7: Develop data sheets

Before any training can take place, data sheets need to be set up for the collection of the measures. These should contain all of the required elements, yet not be too complex to fill in. The purpose of these was to provide a temporary repository for the information over a three-month period, until the measures could be firmed up a bit more, and the collection of data and the reporting of the measures/metrics on a monthly basis could be automated. The data sheets were split into separate sheets for each major process area. This led to a total of 4 sheets, which were amended to suit each project being measured. The following is a brief explanation of what each of the four sheets contains. A detailed explanation of what each sheet contains is in Appendix F.

Release: Containing all software and documentation release information (product, disk and film releases and re-releases), as well as the current product delta

Software Localisation: Containing all effort, productivity and schedule measures for the localisation of the software.

Software Testing: The testing and bug-fixing effort is contained here, along with all bug information (categories, types, and density).

Documentation: This sheet contains all information for the documentation localisation process. This includes effort, productivity and rework measures for the formatting, art preparation and documentation review stages of the process.

These sheets were designed as spreadsheets, which each team leader could fill in for their team at the end of each month. Each project would have its own set of four spreadsheets for each month. The granularity suggested was to record time in mandays, and in order to get a cumulative figure, ie total-to-date, the spreadsheets for each month could be consolidated together, and the corresponding cell in each month's spreadsheet summed together.

The four spreadsheets were provided to each department, and they could arrange their own directory structure and metrics collection/reporting plan.

6.8: Provide necessary training

Workshops

To get full involvement and to ensure that everyone had the same interpretation of the metrics, workshops were held for those responsible for the collection of the metrics. A half-day workshop-type course was developed, and made available on request. The following paragraphs give a brief outline of the course content.

Each person got a copy of the metrics user guide and the data sheets. The session commenced with a very brief presentation on why metrics are necessary for process improvement, and the benefits of measuring for each person in the group present.

It was stressed that the metrics were for their own use as a team primarily, and it was not something they had to do primarily for someone else. The information gathered would be used by those who collected the metrics to improve the projects that they themselves work on. A second reason for the metrics system is to demonstrate process improvements to superiors in the US. Metrics quantify the benefits of any process changes.

After the presentation, the group in attendance was split into two groups, and each of the four data sheets was studied in detail, with one group acting out the role of the unconvinced, who didn't want the extra work involved. They were to find the holes in the system, ie act as though they were opposed to the system. The other group had to defend the proposed measures, and sell the benefits to the cynics.

The result of the workshops was that those who attended felt they had a say in the system, and some changes were made to the definitions of measures as a result of the workshops.

Users guide

The users guide was an attempt at matching the data sheets to the metrics to be collected, and the benefits that each measure would give. It starts by stating the need for metrics and the reasons behind the introduction of a company-wide system. It briefly explains each metrics category, then goes into a detailed table covering each measure. The benefits/trends column in Appendix F explains the usefulness of each of the measures.

6.9: Measure current process and products

Before future improvement goals could be set, the current processes had to be measured. This involved getting estimates of measures for projects just completed. It was decided that 80% accuracy would be good enough, for the purpose of identifying trends, as further accuracy would take much longer to collect. The measures of interest were primarily productivity, rework and quality measures.

Productivity and rework measures were calculated restrospectively for projects that had just completed. Going back to the previous version of every project was not feasible, as the data was not available for many of these projects. The data obtained was used to form the baseline, so that it was known how long it took to localise 200 dialog boxes in project x, and the percentage of rework on project y. From there, the effort, rework and productivity measures for each project are compared against the measures obtained for the previous version in order to quantify improvements.

Regarding quality measures, bugs have always been reported in a database, as detailed in section 5.1, so this information was not too difficult to collect. Defect densities were calculated for the last version of each product, by counting the number of words, diving this number into the number of localisation bugs reported, and multiplying by 1,000. The bug analysis was a bit more difficult to extract. Unfortunately, the fields that had been used to report the bugs were not what needed to be measured, ie the types were not defined the same for each buglist. It took two full months of work to extract the required category information for 50 bug databases, so that clear comparisons could be made. The data was represented on pareto charts which show the number of bugs in each category in decreasing order, with the cumulative percentage also displayed. The information it provided was very useful, and gave the improvement process a good kickstart, by identifying the areas where most bugs were clustered.

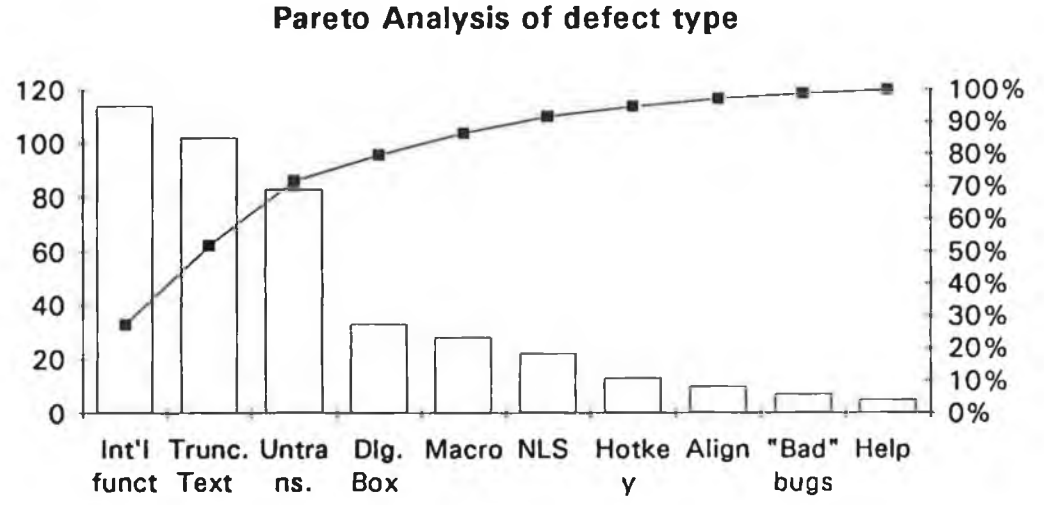
As an example, it was found that there were a lot of duplicate hotkey errors, which could easily be eliminated by developing an automated hotkey checker. On completion of translation of the extracted text files, the localisers themselves now run a hotkey checker on the product, and correct any errors they themselves find, before stating localisation has been completed. First measurements on some projects showed only a 50% improvement, but later projects showed the number of hotkey errors down to almost zero.

Another area found to be very high was untranslated text in error messages, so now all error messages are extracted from the localised product, and the list is browsed through looking for any English text. Again, this check is done before the Localiser states the localisation of each section is complete. As a result, the number of bugs reported against untranslated text has diminished.

If the current process and products had not been measured, to obtain a baseline measure from which the company could set out to improve, managers would not be in a position to claim that they have improved the process in the last year. They would know some improvements had been made, but this is not much use if the improvement cannot be demonstrated in quantifiable terms.

After this exercise, the bug database keywords were subsequently redefined, so that this information, and several other categories of data could be extracted by writing a very simple query, which would report the required numbers within a matter of seconds. Section 7.1.3 b) explains the keywords standardised on.

The following chart shows an example pareto chart for a localised product. It is a typical example of a product which was not developed for global markets - ie there are many international functionality bugs, and truncated text problems.



6.10: Set improvement targets

Once where the process is now was identified, improvement targets in each area could be set. These tied in with the goals set earlier (see section 6.5). The improvement targets were set per project, as the data from each project showed variations in results, and some projects were able to improve in certain areas quicker than others, due to different start dates, methods, tools, experience levels of staff, etc. Some example improvement targets are reproduced here, although these targets have not yet been reached.

1. Reduce testing effort (manmonths) on version B of product X to 50 % of the testing effort on version A through a combination of risk assessment, reduced functionality testing and increased automation of localisation testing.

2. Reduce the bug density on version B of product X to under y defects per thousand words, as compared with z defects per thousand words in version A.

3. Increase the bug per manmonth ratio on version B of product X by 30% as compared with the ratio achieved on version A.

And a general improvement target is to find a higher percentage of the bugs earlier in the process. It's difficult to express in terms of numbers, but can easily be observed by charting the number of bugs found each week, and noting the change to the shape/slope of the chart. (See section 7.1.3 b).

One of the product teams has started working towards goal no. 1 above. The current test plan estimates a 50% reduction in the number of testing manmonths compared with the previous version of the product. This was made possible by analysing each of the metrics for the previous product in detail, and determining where improvements could be made in each area. Improvements include further automation of localisation testing; reducing the amount of core functionality testing, based on the functionality bugs found for the previous version measured against the time spent testing core functionality; proper risk assessment of each product area, and planning test time accordingly, and automating the testing for the most common bug types found in the previous version. The combination of these improvements should show a testing time reduction, fewer test passes required, bugs found earlier in the process, etc. ie goal no. 3 above should also be satisfied.

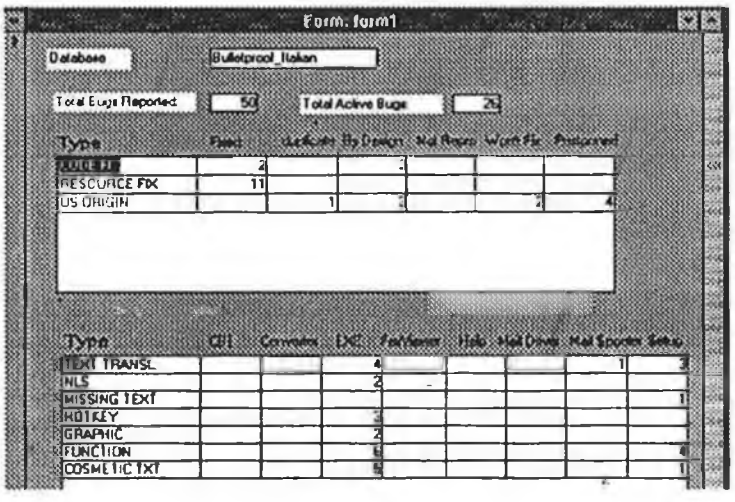
6.11: Automate the system

The metrics system started off as a manual system, as some changes were expected to be made after the first few months. Automation involves setting up a database system for collecting and reporting on the measures. There are really three distinct areas involved in the system.

Firstly, there is a project scheduling system, which is based upon a standard work-breakdown structure (from the process mapping activities). Direct involvement with this initiative was minimal, except for attending one of the meetings to give an overview of schedule metrics. More details of this system are provided, under scheduling metrics, in Section 7.1.2 c).

Secondly, there is an effort-tracking system, which is based on a system developed by a tester on one of the test teams. The format of the system is fairly similar to a system that was in operation about two years ago, but which was too cumbersome to use, and had to be completed by Managers only, which took up a lot of management time unnecessarily. This system was further defined using the work of the metrics team, and the middle managers in the company. The necessary amendments were made to enable it to be used company-wide. A further description can be found under effort metrics in section 7.1.2 a). One of the departments has continued to use the original spreadsheets, and imports the data on a monthly basis to a database, from where reports are generated.

Thirdly, there is a simple database report system to extract the bug analysis data from any bug database, in a matter of seconds:



7: PRESENTATION OF LOCALISATION METRICS

Paragraph 4 describes measures in terms of the categories they belonged to, rather than per process stage, as different process stages may use the same measure which makes it difficult to describe the appropriate measures. However, when defining specific measures to be implemented in a company, this should be done by process stage, and choosing the appropriate measures from chapters 4 and 5. The measures are reported here by process stage, and cross-referenced to metrics described earlier, where appropriate, rather than being repeated in several places.

As already stated, in the introduction to Chapter 6, the localisation process assumes the coding of the product has already been completed, and is primarily concerned with changing the user interface, to look as if the product was developed for each specific country in which it is sold. The project development phases in a localisation environment are different to the traditional phases, although there is a strong similarity.

For this reason, the traditional size and effort metrics, eg. Halstead's Software Complexity Measure, McCabe's Cyclomatic Complexity Metric and Boehm's COnstructive COst MOdel were not used in the case study.

The measures described here are an initial set of measures, which were introduced in the company. They were introduced in order to control the process, monitor the costs, and then to provide information as to areas that needed improvement. Some projects/departments have implemented more of the measures than others, as the company is too large (400+), and the projects too diverse (home products to advanced network systems) to implement all measures at the one time. Typically, measurement is introduced as new projects start, and projects that were already in progress when metrics were introduced just report end-of-project measures, as in-process measures would not have been worthwhile to introduce half-way through.

Appendix F gives a full explanation of each of the measures/metrics originally suggested. Appendix H shows the minimum set of measures currently gathered, 6 months later.

The phases identified in section 6.1 were:

Software	*Description*
Planning/preparation	involves preparing localisation and test plans; setting up the localisation compile kit; extracting files and strings to be localised, preparing schedules.
Localisation	of the main program; the helpfiles; the Computer Based Training program; the Setup program, and any other addins.
Testing	of all components that have been localised and recompiled.
Bug Fixing	of all bugs found during testing.
Releasing	of disks to manufacturing for mass duplication.

Documentation	*Description*
Planning/preparation	vendor selection; preparing documentation for translation; preparing schedules.
Translation	of the documentation by the vendors
Formatting	of the translated documentation
Art Preparation	taking screendumps of the localised software, and placing into the translated documentation
Review	of the documentation - translation, formatting and artwork
Release	of film to the printing Vendor via Manufacturing.

Taking each phase as above, the applicable metrics that were selected in each case are described in detail in the following sections.

7.1 SOFTWARE

7.1.1 Planning

The highest level planning metric is the measure of the time difference between the release of the US product and the release of the localised product. This number is expressed in the number of elapsed calendar days, referred to as the 'delta'. A delta of zero for the major languages of all main products is the ideal situation. Rather than stating a specific release date at the start of the localisation effort, the delta is published. This allows for any changes in the US release date, which has a knock-on effect on the release date for the localised versions. (i.e. the release date may change due to circumstances outside the team's control, but the delta must be met).

Release Goals are set using the 'delta' as the measurement. A goal may be to release the first five language versions with a delta of 30 days, the next five with a delta of 60 days, and so on, until a required delta has been set for each language.

Measuring the current delta against the published delta tells whether or not the team is on target. Measuring and reporting the deltas gives the ability to work out the average delta for each language version of the products across the Product Unit or across the company. The sales subsidiary in each country plans its marketing campaigns around the expected delta for each new product.

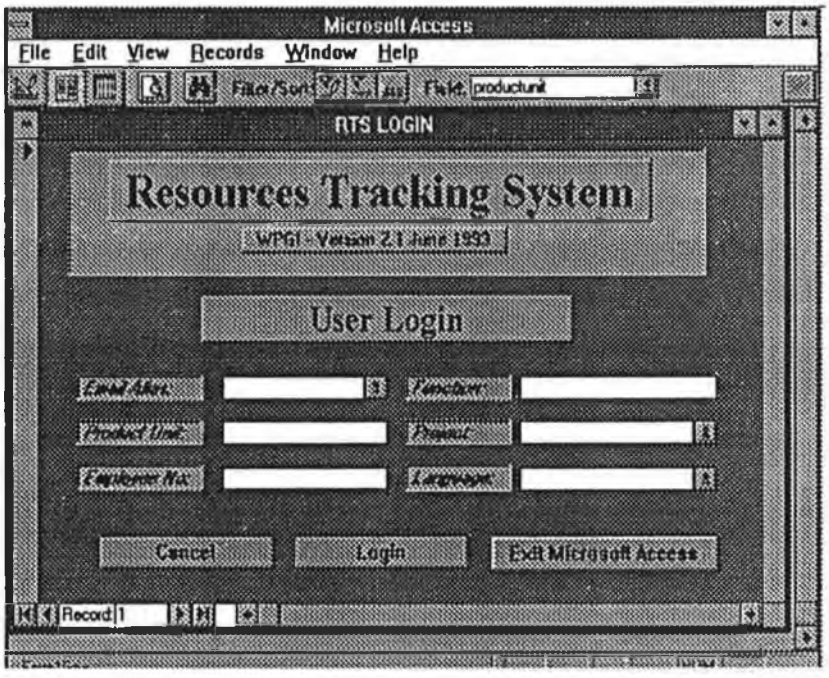
Accompanying the delta is the expected release date for each version of the product (starting with the US product). The release date is given a confidence factor, which helps the planning of resources for projects and the planning of product launch exhibitions. With a confidence factor of High, (-1/+1 week), it is fine to plan a marketing campaign, or plan for the people working on the project to start a new project, the week after the expected release date. Whereas, a confidence factor of Low (-0/+4 weeks) gives the likelihood of the expected release date to slip by up to 4 weeks, so it would be unwise to base important scheduling decisions on the published release date.

All of these measures are recorded in a spreadsheet which is a very top-level view of the schedule for each project in each of the Product Units. This spreadsheet is updated each fortnight, and contains basic information such as the product name, language, version, project start date, release to manufacturing (current vs. planned), shipdate (current vs. planned), the delta and the confidence factor.

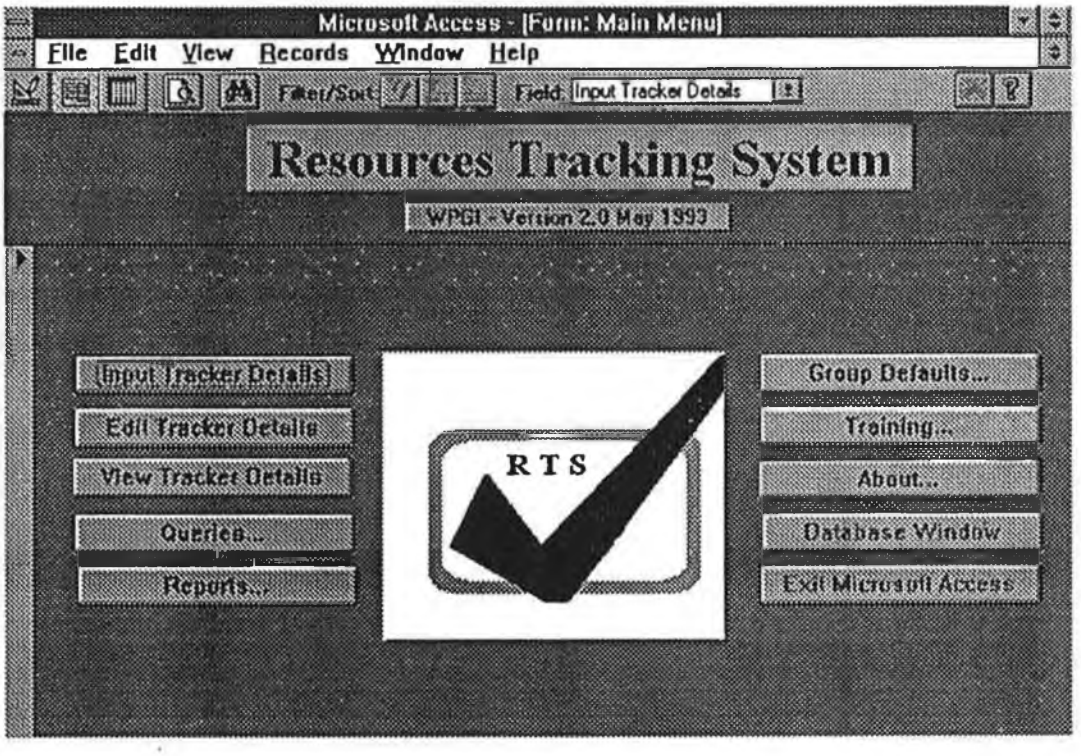
7.1.2 Localisation:

A). EFFORT:

The number of manweeks of effort spent localising each section of the software is measured. The software was divided into several elements, including the localisable strings and dialog boxes in the main program, the Computer Based Training modules, the helpfiles, and any add-ins included in the product (macros, sample files, etc.). Effort is recorded at the individual level, in a database. The database system tracks effort for all functions, and is currently being implemented company-wide. This database was originally developed by a Tester in one of the Product Units, and has subsequently been expanded to cater for all other functions.



This system has a series of forms and tables, in which the required information is entered and stored. After a successful login, the user sees the main menu, where the desired option can be selected. Different task entry forms will appear, depending on the function of the person entering the data. Each functional screen has its own set of categories against which to enter time information. There have been several iterations of what 'should' be included in the effort tracking system. Appendix G contains the most recent set of tasks against which effort is measured.



Both Cumulative and Non-Cumulative Effort distribution can be calculated and charted, from the reports printed from this system. The time period to use would be by month, as weekly reporting and charting would take up too much time, and is not necessary.

Effort tracking is also tied in with cost tracking. The effort recorded in persondays is multiplied by the average cost for each function, to get the project cost information, which is reported to the parent company in the US.

B). PRODUCTIVITY:

The quantity of items localised was recorded, ie project size, and divided by the number of manweeks of effort, to get the productivity rates. The number of strings and dialog boxes localised, the number of CBT screens and the number of help pages were counted, and the effort expended on each of these activities. All add-ins are different, and cannot be quantified, hence one can only measure the time taken to complete each add-in, and not productivity as such.

These productivity measures are reported at the end of the project, rather than throughout the project on a weekly or monthly basis. Frequent monitoring throughout the project lifecycle would be very useful, and would help identify potential problems before they occur. However, this is a goal for a future expanded implementation of metrics. A good metrics infrastructure should be in place first.

- **Strings translated**: The number of strings translated divided by effort spent by the Localiser. This gives a ball-park figure of how long it takes to localise files of extracted strings, and will measure the efficiency of new tools as compared to a fully manual process. The number of dialog boxes prepared and effort spent was originally suggested as a separate category to strings translated. Software Localisers spend most of their time either translating tokens or sizing dialog boxes, but it was later decided that it should be considered as all the one activity, to start with.

- **CBT Lessons**: The number of CBT screens localised divided by effort spent. This shows throughput for the project. Because different development methodologies are used for CBTs across different projects, straight productivity rate comparisons are not feasible, but this metric will demonstrate any major differences in productivity due to the differing tools/methods used.

- **Help Pages**: The number of pages of Help prepared divided by effort spent.

- **Add-ins**: The time taken to localise each add-in. Add-ins cannot be measured in quantity in the same way as the other elements, but can take up a substantial amount of time to complete, and therefore should be thoroughly planned for. A productivity measure would be the number and type of add-in that can be localised in a manmonth.

C). SCHEDULE:

One of the process improvement teams took the process map, and made it into a standard work-breakdown structure, which is used as a template for scheduling localisation projects. This system was developed using Microsoft's project planning product, MS Project 3.0 for Windows. It consists of several different modules - Create a Schedule; Maintain a Schedule; View a Schedule and Print a Report.

Create a Schedule

This module is used to create a new schedule or to make extensive changes to an existing schedule. This module uses the template containing the standard work-breakdown structure, including generic project milestones, from which to build schedules.

Maintain a Schedule

This module tracks the project as it progresses. Tracking data for all incomplete tasks can be entered, and the status of all milestone dates viewed. It will give the current status of each task, % complete, etc. as a Gantt chart. This module is the most useful for the metrics purposes.

View a Schedule

This module allows various views of the schedule data, with read-only access. Critical tasks can be viewed, tasks assigned to a particular person, different levels of tasks - milestones, top level tasks, all tasks, etc.

Print a Report

This modules has options to print a variety of reports. Again, most interesting for the schedule metrics is the Gantt chart showing actual vs. planned completion by task. Other reports include the PERT network, critical tasks, tasks assigned to a particular person, milestones, full task list, etc.

At the beginning of each project, the total expected quantity of each item is stated, and as the project progresses, the cumulative effort and amount completed to date is known. From the % complete figure, and using the productivity metrics, the remaining tasks can be scheduled effectively.

7.1.3 Software Testing

A). EFFORT:

This is measured in terms of mandays per area/task, and the number of test passes per area. The areas that are defined include the main executable, the setup program, the CBT, the helpfiles and the add-ins (such as macros, 3rd party executables, sample files, etc.) Another area that is measured (primarily in the bug reporting database) is the time involved in fixing the bugs found. The Resource Tracking system has already been explained. The following table gives the main categories used - there were some other categories, such as 'meetings' and 'other', which are not particularly useful, and will be deleted for future implementations.

Main Exe	Add-ins
Project preparation	Setup
CBT	Wizards
Bug Fixing (Engineering)	Help
Cue cards	Tools/Research

Note that 'bug-fixing' is listed as a separate phase in the table at the start of chapter 7, but is included here, as only effort expended on bugfixing is measured, and there is no need for a separate section to discuss this measure.

B). QUALITY:

Section 5.1, defect reporting, explains the need for a bug-reporting system, and briefly describe the fields such a system could contain. It mentions that there are six keywords, which may be amended by the Test Manager for each project. For localisation projects, the values contained within four of these keywords are standardised for Ireland. The first two keywords, Main Area and Sub Area are defined by the US Test Manager, and often are known by different titles too. The other four keywords are described here.

Definition of 'Type'

The 'type' keyword describes the category of the reported bug. There are twelve categories in total defined for this keyword, of which the main options are explained here:

Text Errors

There are two sub-categories of text bug - cosmetic anomalies, and translation errors. The cosmetic option covers: missing text; incorrectly displayed text; incorrect formatting or alignment, and truncated text. Translation errors include untranslated text, incorrectly translated text, mixed languages, spelling and punctuation errors.

Dlg. Box Errors

This covers any misalignment, missizing or mispositioning errors concerning either dialog boxes or their components (check boxes, buttons etc.).

Hotkey Errors

These include missing, inconsistent or duplicate hotkeys.

Macro Errors

This option covers macro recording problems, and errors in the supplied sample macros.

Functional Errors

These are further broken down into sub-categories such as crash/hang bugs, problems with National Language Support (ie incorrect sort order, date format, etc.), and product functions that will not work

Help Errors

Many errors found in the help files can be classified under the other 'type' options, with the exception of help jump/popup errors, which is another option within the 'type' keyword. Within helpfiles, one should be able to select a topic, and get the correct topic displayed on the screen, ie it will 'jump' to the topic. A popup is a small screen accessible from selecting a highlighted word within a topic which explains the word or phrase.

Definition of 'Origin'

Origin Values for this keyword are fixed and are comprised of the following:

CODE FIX	International bug requiring source code fix.
COMPILE ERR.	Bug that occurred due to an error in the compile process.
RESOURCE FIX	Localisation bug requiring resource/token file fix.
US ORIGIN	Bug that affects the functionality of the US product.

Definition of 'Fix Time'

This keyword is filled in when the bug has been resolved. It applies to the amount of time it took to solve and verify a fix. It refers to the time expended on the bug, rather than the elapsed time. From a metrics point of view, this keyword is used to get a profile of approximately how long it takes to fix each category (or severity) of bug, rather than obtaining exact time information. The options under this keyword are: < 1 hour; 1 - 4 hours; 4 - 8 hours; > 1 day, and > 1 week.

Definition of Keyword 6

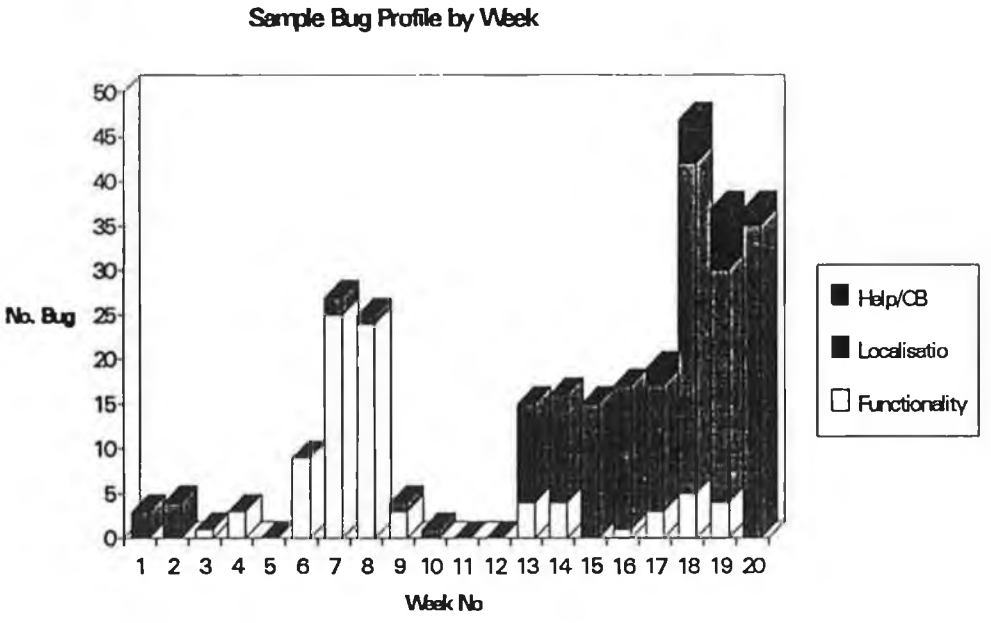
The sixth keyword definition is left to the discretion of the test team. Some teams use it to record the Test Case # which was used to uncover the bug, and some other teams, working on cross-platform products, use it to record the platform on which the bug occurs (ie options are Win, Mac, DOS, and All).

TOTAL NUMBER OF BUGS:

This is the total number of bugs in the bug database on the last day of each month. This gives an indication of the quality of the product, since the fewer bugs the better. Number of additional bugs per month should decrease as the shipdate approaches. The total bugs per language should decrease as the releases proceed. Improving the localisation process should produce fewer total bugs from one release to the next.

BUG FIND RATE

The bug find rate shows the number of new bugs entered into the buglist each week. A typical chart should show the total number of bugs start off low, increase rapidly, form a plateau, then decrease rapidly towards release (see section 5.2 d)). Functionality bugs will be found first, and the reason for the slow start is that some functionality bugs must be fixed before parts of the software can be tested, and this pattern can also be due to a relatively clean first handoff of software, with a large, buggy handoff coming several weeks later. The plateau occurs at the optimum bug finding/reporting rate. This depends on the number of people on the team, and the average time it takes to find and report an obvious bug. The rapid decline towards the end occurs when the software is relatively defect-free. This chart should be updated each week by the Test Team, to show progress.



The graph above shows the status of the bug database over a twenty week period, with the weeks numbered consecutively from 1 to 20. Looking at the graph, it can be deduced that the software is not yet ready to release - large numbers of localisation 'user-interface' type bugs are still being found, although the pattern for functionality bugs shows that the product is functionally stable. Localisation and help/CBT bugs are easy and very quick to fix relative to functionality bugs, and so if week 20 of the localisation testing effort has just completed, and week 21 has started which, say, shows a sharp decline in the number of new bugs reported, then the estimate is that the software will be released in about three weeks' time.

NUMBER OF ACTIVE BUGS:

The number of active bugs, as compared with the total number gives an indication of how near to release the product is. The age of active bugs is also an indication of the stability of the product. This is easy to calculate by subtracting the 'opened date' from the 'resolved date', and expressing in days. Categorise the bugs according to the numbers of bugs of each 'age'.

BUGS PER MANMONTH (MANDAY):

The number of bugs per manmonth (manday) shows the cost of testing the product - the higher the number, the better the testing methods. This number on its own can be a bit misleading, so the total number of bugs must also be looked at (ie a really bad quality localisation may also give a high number, but may not necessarily mean that the testing methods are wonderful). A very good quality product with few bugs should take less time to test, and fewer test passes, hence the number should still be high. The bugs per manmonth number is obtained by dividing the total bugs by the total testing effort (in manmonths/mandays). The total number of bugs are all those in the database, excluding 'duplicate', 'by design' and 'not reproducible' bugs. The total manmonths includes time spent testing the product, and excludes all testing preparation/familiarisation/training time.

CATEGORISATION OF BUG ORIGIN:

In the localisation of software, there are four bug categories, the three main categories are: those found in the original product, and are not as a result of the localisation process; those that result from trying to localise the product, but which may need some source code changes to fix them (ie they also originate with the US product, but are only an issue with the localised product - eg. not supporting a country's date format or its sorting order); and those that are introduced during the localisation process itself - text alignment, untranslated text, duplicate hotkeys. The fourth category is bugs due to a compile/build error, ie pressing the F1 function key to access the Help no longer works in the current build.

- **US bugs:** Shows the number of bugs which also occur in the US version.
- **Code Fix**: Gives a measure of the localisability bugs remaining in the code.
- **Resource fix**: Shows the number of bugs introduced during the localisation process. Helps focus the improvement efforts on the main causes of these errors. The number of localisation bugs should decrease as the process improves.
- **Compile Error:** Shows the bugs due to errors in the compile process

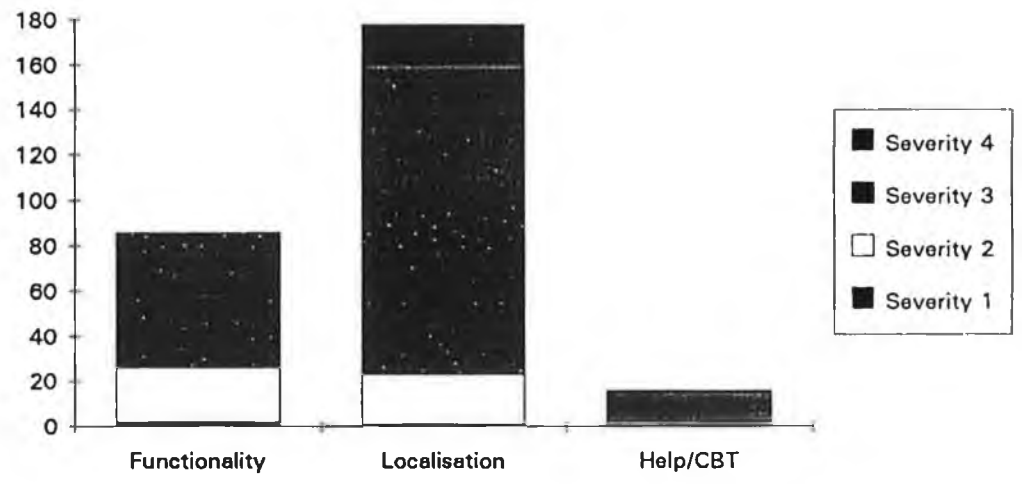
BUG DENSITY (BUGS PER KLOC)

To find out the quality of the localisation effort, bug densities were measured. Instead of using Lines of Code, localisable strings and words were used. The principal is the same. Bug Density is the total number of bugs in the database divided by the size of the product in words, expressed per thousand words. It is calculated at the end of the project, and gives a good indicator of the quality of the localisation. The total number of bugs includes all localisation bugs and international functionality bugs, but excludes bugs of US Origin, ie those bugs that also impede on the functionality of the US product. Also excluded are duplicate bugs and those that are resolved as 'by design' or 'not reproducible'.

This measure is great for comparing the quality of two different products, of totally different size. It is also excellent for determining the relative quality of product upgrades. eg is version 3.0 of a product has a total of 200 bugs, with 20,000 words, the density is 10 bugs per thousand words. If version 3.2, 9 months later, had 60 bugs, then on the surface the quality might seem fine. However version 3.2 might only have involved the localisation of 4,000 words, which is a defect density of 15 bugs per thousand words, which is of worse quality than version 3.0.

BUG SEVERITY

This gives a distribution of bug severities across different modules or by bug type, as below. Most localisation bugs fall into the severity 3 or severity 4 category, whereas functionality bugs are more likely to be severity 2 and severity 3, and occasionally severity 1. This graph can also be depicted in terms of the % of the total bugs in each category.



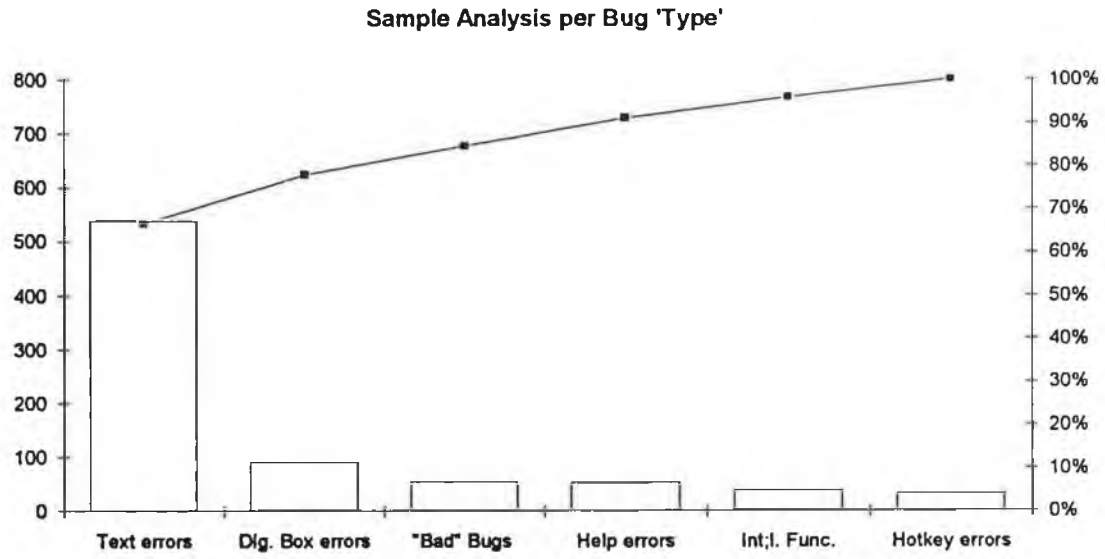
TEST PASSES:

The number of test passes gives an indication of the quality of the software (the higher the number of test passes, the poorer the quality of the localised software.) In an ideal world, there would be 2 test passes (the first one and the regression pass). The number of test passes for each test area should be monitored monthly, and reported at the end of the project. The total number of bugs per test pass can also be measured, which is has similar uses to the bugs per manmonth metric.

TYPES OF LOCALISATION ERROR:

Because Microsoft Ireland's business is the localisation of products, the localisation bugs needed to be further classified. The categories chosen were text bugs, hotkey bugs, dialog box errors and help jumps to start with. Analysis of current projects is in accordance with the definition of the 'type' keyword, in section 7.1.3 (b).

This shows the type of bugs present in the software, and their relative percentages. It helps to concentrate effort on localising the next version of the product in certain areas to eliminate the most common bug types, and helps indicate where testing effort should be spent. The chart below shows that the team should concentrate on eliminating text errors for the next project.



7.1.4 Releasing

QUALITY METRICS

Ideally, the products should be released once, and not re-released as a result of bugs introduced into the product during the localisation process. This is the measure that the subsidiaries rate the project team on, and that which is most visible. Ideally the number of product and disk releases per month should increase, whilst the number of re-releases should be zero.

The measures chosen on the software side were the number of product re-releases as a percentage of the releases, and the number of disk re-releases as a percentage of the number of disks released. These metrics needed precise definition, to avoid confusion and ambiguity.

Measuring the number of product and disk releases and re-releases will give an indication of the amount of rework involved after release. Correcting errors at this stage costs infinitely more than finding and correcting them on the first test pass.

The release measures, reported in monthly status reports, are:

- **Product Releases:** The number of product releases this month
- **Product Re-releases:** The number of product re-releases this month (ie new part numbers generated)
- **Disk Releases:** The number of disks released this month
- **Disk Re-releases:** The number of disks re-released this month

A product is released when the paperwork is completed, signed off by the relevant people, and ready for duplication in the manufacturing facility. A product re-release occurs when a new part number is required for any disk due to a change in the software contained on that disk.

A disk is released when a copy of the disk image is transferred to the Trace duplication system in Manufacturing. A disk is re-released if, for any reason a disk has to be resent to Manufacturing, necessitating a Master Disk Control number change, whether or not the software on the disk has changed. A disk re-release is counted *before* the product is signed off, whereas a disk re-released *after* the product is signed off also constitutes a product re-release, as defined above.

7.2 DOCUMENTATION

The documentation process lends itself relatively easily to measurement, as the process is easy to define, and quantities can easily be counted, in terms of words or pages. There are no specific planning measures in documentation - the planning measures explained in section 7.1.1, such as the 'delta', are also relevant here.

7.2.1 TRANSLATION

Translation of the documentation is the responsibility of the translation vendors, so the measures defined here are a measure of the quality of their translation. At a later date, the vendors should provide information on productivity metrics.

A). REWORK:

It was decided to measure rework that is sent out to the translators, as traditionally this has been very high. The reason for this is the large number of updates coming from the technical writers who develop the US documentation. These changes are sometimes functional changes that must be implemented, other times there are cosmetic changes that make a sentence more aesthetically pleasing, which are also included in these updates. The measure decided on was the number of pages that have changes on them that are sent to the translators.

B). QUALITY:

Quality Assurance spotchecks are performed on the documentation (about 10% of it is checked), with error counts recorded and compared against 'acceptable' criteria that have been defined.

- The translated documentation must be a clear and technically correct description of the actual functionality of the localized software. The translated documentation must contain culturally adequate adaptation of names, locations, scenarios etc., and must adhere to technical and country specifications. For Help and CBT also: Layout and functionality of translation must be equivalent to the US version.
- The translated documentation must be completed by overtyping without adjusting appearance of text, removing or changing in any way the paragraph formatting or layout of the document or deleting any elements, e.g. index codes, art references, etc.

- The translated documentation must be complete as to its contents, adhering to the Microsoft Linguistic Guidelines, and linguistically correct according to recognised rules and recommendations.

Rating A 1-to-5 rating scale, in 0.5 increments, is used for the Vendors. The goal is to achieve a quality level of 3.0 or higher for all components, and in all categories.

The global rating definitions are:

5	**Exceptional quality, no errors**
4	**Excellent quality, very few errors**
3	**Good quality, few errors**
2	**Needs improvement, many errors**
1	**Unsatisfactory, many serious errors**

Further details of the rating system and categories used are in Appendix E.

C). LOCALISER REVIEW

When the documentation comes back from the translation vendor, the localiser performs a technical review, to ensure that the software and the documentation match eachother. At this stage, they often find errors made by the translation vendor, which they correct. The number of pages that have to be reworked as a result of translation error is therefore measured. The second rework category is due to updates from the US, which has already been discussed. Another factor of rework is rework due to localiser error. Again, this is measured in terms of the number of pages affected.

Doc rework: - should decrease and approach zero

2. **Translate:** No. pages reworked due to errors introduced by the translations vendors.
3. **Update:** No. pages reworked by the Localiser due to updates from the US group.
4. **Loc:** No. pages reworked by the Localiser due to Localisation error.

7.2.2 Formatting

The editors format the localised documentation in a standard style, and generate the index for each manual. The documentation consists of several manuals, covers, flyers, etc.

A). EFFORT:

It was decided to measure the amount of effort spent on formatting the documentation in terms of mandays/weeks per month. This is also recorded in the Resource Tracking System as described in section 7.1.2.

B). PRODUCTIVITY:

The number of pages of documentation formatted is divided by the number of manweeks of effort, to get the number of pages per manweek. This is completed at the end of the project, rather than by month throughout the project.

In order to calculate productivity, the following are calculated at each project-end:

- **Resources:** The number of each function who worked on the documentation.
- **Pgs. formatted:** The number of pages formatted by the Editor(s) and the total effort spent.
- **Pgs. reworked:** The number of pages reworked and the effort spent.
- **Loc QA:** The number of pages QA'd by the localiser, and the effort involved.
- **PTL QA:** The number of pages QA'd by the Production Team Lead, and effort.

C) SCHEDULE:

At the beginning of each project, the total expected quantity of pages to be formatted is stated, and as progress is made through the project, the cumulative effort and amount completed to date is known. See section 7.1.2 c) for a description of the schedule measures.

D) REWORK:

Formatting rework is due to errors made by the editor when formatting the documentation, or can be due to changes that the translation vendor made to the formatting or hidden codes. The number of pages reworked as a result of formatting errors was measured.

7.2.3 Art Preparation

Art Preparation involves shooting screen-dumps of the localised product, and preparing call-outs, then linking the art to the appropriate places in the manuals. The measures originally suggested included art as a separate entity to documentation, but since it is an integral part of the documentation process, it was later decided to merge art with documentation for metrics purposes. The following paragraphs explain the measures originally agreed for the art.

A) EFFORT:

As with all other phases, the effort involved is recorded in the Resource Tracking System database.

B) PRODUCTIVITY:

The number of screendumps shot and the number of pieces of art prepared divided by the respective number of manweeks, gives the average throughput per manweek.

C) REWORK:

Art rework is measured in a similar way to documentation rework, according to its causes. Again, there is rework due to updates from the US, and then there is internal rework - that due to localisation error, leading to screen dumps having to be reshot, and rework as a result of formatting error during the preparation of the art by the artists. Both the quantity of rework and the effort involved are recorded.

Art rework: - should decrease and approach zero

1. **Format:** No. pieces reworked by the Artist due to formatting errors.
2. **Update:** No. pieces reworked by the Artist due to updates from the US.
3. **Loc:** No. pieces that were reshot due to errors in the original dump.

7.2.4 Release

The documentation is released when the files have been handed over to the print production group, to be sent out to the printing vendors. A re-release of documentation occurs when new files or film is sent to the printing vendor. The number of pages that have been re-released and the severity of the error also need to be recorded

7.3 Summary

The following is a summary of the implementation phases for localisation metrics. The first column contains the minimum measures that should be implemented on a project, ie those started before the metrics were implemented. The second column represents the full initial set of measures, and the third column represents the set of measures that should be implemented in the next phase of implementation.

Category	*Minimum*	*Full initial System*	*Next Phase*
Size/ Complexity	Localisable Words	Localisable Words	Localisable Words with complexity factor for menus, dialog boxes, etc.
Productivity	Productivity rates per function - end of project	Actual vs. planned productivity rates per function per phase -end of project	Actual vs. planned per function/phase within approved range, measured monthly
Rework	Total rework quantity - end of project	Rework quantity and effort expressed as a % of total measured quarterly	Rework effort and quantity per cause category and as % of total effort, monthly
Effort	Total effort per function - end of project	Per function per area (actual vs. Planned) - end of project Effort distribution per task (monthly)	Fully automated effort tracking and reporting system - monthly
Schedule	Delta Simple Gantt chart	Delta Gantt chart with % complete against generic milestones, on monthly basis	Delta Progress curves categorisation/ cumulative work packages complete, throughout project
Quality	Total defects Defect density Pareto analysis of type Categorisation of origin	Defect find rate profile by week No. active bugs by week Pareto analysis of type Categorisation of origin Defect density Defect severity Bugs found per testing manmonth	Software reliability measures Defect find rate profile (by day) Pareto analysis of cause Age of defects Other Quality measures same as full initial system

8. ANALYSIS OF RESULTS

8.A. - Process

The 12-step process was followed, which went quite well. The results of each step in the metrics implementation process are analysed in the following pages. The step which gave the most payback was to measure current products and processes. Probably the largest area for improvement is in the effort tracking area, in that too much was measured too soon, despite knowing and realising that this was a pitfall. Keeping track of the plans for implementation of measures on all projects was difficult, due to the size of the company and the number of active projects, if each language version is considered as a separate project. Implementation of the metrics should have been on a pilot project over a duration of three months, or on just one project in each Department (Product Unit) rather than a company-wide implementation.

8.A.1 - Map the software development process

As explained in section 6.1, several other process improvement groups had defined the current software process, which could be leveraged off. Process charting is an excellent method to gain full understanding of the process, and the resultant chart should be used as the basis for effort tracking. There are a few important learning points for this activity:

- For initial metrics purposes (which will be relatively high level to start with), a fairly high-level process map will suffice. ie too much time should not be spent going into minute detail. The process map should contain about 30 steps in total, covering all functions.

- Different projects use different tools for certain process steps, so it is important to map what happens (the process), rather than how each step is performed (the methods).

- Make the map simple, and easy to upgrade, as it will change as improvements are made.

- Display the process where it will be visible to all employees, and keep it updated, so that progress in process improvement is visually demonstrated.

8.A.2: Define the Corporate Improvement Goal

Section 6.2, states that the company had a generic goal, to improve or processes in term of time, cost and quality, and a more specific longterm goal, to consistently release 20 products in 60 days within a given timeframe. These goals were defined by the CEO, and explained to all employees at a company meeting.

The two important issues here are

- Keep the goal longterm (as in the ultimate goal of the improvement efforts)
- Establish a measurable goal (in order to measure progress as time goes by)

Implementation of the metrics system would have been smoother, and less isolated if there was a full Quality Management System (QMS) tied in with the Corporate improvement goal. This includes the qualitative cultural activities as well as the quantitative metrics activities. Nevertheless, in order to reach the longterm goal, the metrics that were firstly introduced were to control the projects, it is only then that a more comprehensive umbrella QMS becomes necessary, to provide company-wide direction.

8.A.3: Conduct an Employee and a Customer survey

These surveys were excellent for gathering current perceptions. In order to improve, everyone must be involved, and to solicit their feedback on the process is an important first step to establishing the areas that are highest priority for improvement. A few points on the employee survey are:

- It was kept anonymous, so that honest feedback was given.
- The survey should be performed once a year, to measure the perception of improvement
- All those who responded to the survey received a copy of the survey results/analysis
- Measures were introduced in the process for the areas that the survey results showed needed most improvement.

One area noted was that under the 'comments' section of the form, some people wrote comments in the form of 'I often fill out survey forms, and never hear anything back. Will we see the results of this one?' It is important that people do not view it as another survey that will go into the black hole.

On the customer survey, the feedback form (see Appendix B) is sent to each Subsidiary three months after releasing each product. This highlights any issues that need to be addressed either for the next version of that product (ie from the product ratings), or for the communication/project management processes in general (from the process ratings). Again, the subsidiary must feel that their ratings and comments will be heeded, and some positive actions will be taken to improve the areas with low scores.

8.A.4: Define applicable metrics categories

These were really easy to define. From the process charts, it is obvious that there has been a lot of rework, so rework was one of the categories.

A lot of time is spent testing a product, and then the next version of the product in the same manner, without analysing the number and types of defects, their causes, and the improvement from one version to the next. Improvement was required in many areas, but there was no method of prioritising these. Therefore Quality metrics were needed.

Project durations, staffing requirements, etc. are initially planned over a year in advance of the start of the project, but traditionally guesstimates of the productivity rates have been used. Effort estimates were often increased linearly with the perceived size of the product. ie there will be 50% extra features, so both localisation and testing should be increased by 50% also. Documentation estimates were a bit easier to devise, as the sizes are generally known when calculating the durations and staff requirements. To enable accurate cost estimates and better project management, it was decided that a combination of size, effort and productivity measures was needed.

One other area where improvement was necessary, which was made obvious by the process surveys was project scheduling, and resource planning. Hence, some schedule measures were included.

These measurement categories, (size, productivity, effort, rework, schedule and quality) are those that were suggested in section 3.4.1, which I believe are fully adequate for an initial metrics system. Results to date demonstrate that these categories are adequate for the company's needs.

8.A.5: Break corporate goal into a specific goal for each category

The corporate goal has been broken down into more specific goals, which are disseminated throughout the company to the appropriate people responsible for achieving them. I involved myself in this step up to the stage where there were enough goals to define some specific measures, and then handed responsibility over to the Product Units. Care had to be taken to ensure that it did not become a system that I was introducing, rather than a system that all Managers understood, needed and wanted. As already discussed (section 8.A.2, above), an overall Quality philosophy would have made the system more cohesive.

Each Manager has a set of goals, which can be broken down further to provide goals for their staff, and so on down the chain, so that each person has his/her own set of goals (ie quality, productivity, schedule goals), which together tie in with the corporate improvement goal. Some examples of management goals are described in section 6.5.

On the down side, the complete picture of the goals for each category are not necessarily known by everyone, ie everyone is aware of their own goals, and their team goals, but it would be nice to show how the goals of each functional team fits together, which is important for a co-ordinated process improvement effort. The main result of this is that the reason for introducing the metrics sometimes loses some focus and understanding, and it's back to the situation of measuring because someone says so, rather than for oneself, and the team.

8.A.6: Design specific measures

This step took up a lot of time, as it involved analysing the specific goals, asking the relevant questions, and defining the measures. Other sections of this document state that one should not try to implement too many metrics at once. Despite this, there was still what would be considered as too much to measure, in the effort metrics category, at the first attempt. However, this fact was recognised quite early on, and rather than say 'no, that's far too_much', it was left up to each department to decide on the granularity of implementation, ie collection/reporting weekly, monthly, or project-end; and using mandays, manweeks or manmonths. As long as the high-level measures were available, no-one minds if a further level of granularity was desired by some Managers. In this way, each Department

understands what's needed and what's superfluous, themselves. At the time of writing, everyone realises that the extra granularity is more of a hindrance than a help, and the company is in the process of moving back to measuring effort at the highest process level, see Appendix G.

In order to transfer ownership to the groups who are directly responsible for completing their projects, effort was concentrated on defining what should be measured, in order to help the process improvement efforts, rather than how they should be collected. Hence, different Departments tried reporting the metrics in different ways, ie some expressed their measures in terms of manmonths at the end of the project, and others measured progress per month, throughout the duration of the project. Each Department adapted the datasheets to suit their projects, and their metrics implementation method. The message to each Department was that for initial metrics implementation, the numbers do not have to be 100% correct. 80% accuracy should suffice to show trends initially, in order to show areas where improvement is most necessary, and to obtain estimates for project planning and cost projections.

The most difficult measures were the progress and productivity measures which it was originally suggested should be reported at the end of each month, throughout the project, for software localisation. The problems were due to the lack of tools that could easily and accurately count the number of dialog boxes, menus, etc. that were localised in the last month. Instead, most projects opted for measuring productivity at the end of each project. Section 8.B describes the implementation of each measure in more detail.

8.A.7: Develop data sheets

The initial data sheets were designed so that the data could be collected, for later insertion into an automated database system. These data sheets included all the initial recommended measures, and were to be completed each month for a period of about four months, until the system was reviewed and automated.

It was left up to each Department to amend these sheets to suit their requirements, and stressed that the granularity of measures was their own decision, based on what they thought was necessary to enable good decisions to be made, and areas for improvement to be highlighted. Each project had its own set of sheets.

One criticism of the sheets is that they were designed in accordance with the main process areas, rather than by distinct function (see initial documentation, Appendix F). If they were to be redesigned now, they would probably be produced according to the person who is responsible for ensuring the data is collected (ie a sheet for each Team Lead). The main problem with the sheets was that each sheet had input from several people, and each person had to input to several sheets. However, this problem was overcome when the sheets were adapted for each project by each of the departments. This approach encouraged ownership of the metrics by each department, as if everything was handed to them ready to go, the need for metrics would not have been internalised.

Whether or not this was an optimal approach is questionable. If a full Quality Management System had been put in place at the same time, with all the cultural philosophies normally associated with it, the metrics implementation would have been a bit more cohesive.

8.A.8: Provide necessary training

Section 6.8 briefly outlines the training workshop developed. This was made available company-wide, but the training was not delivered as widely as necessary - the leads/managers from three departments (out of six) received the training.

If the company was to start over again with implementing a metrics system, the availability of the course would be advertised more, and it would be compulsory that all team leads and managers attend. People do not fully understand the sometimes subtle difference between measures (raw data) and metrics. An example of data would be a total number of bugs reported, whereas a metric would be the number of bugs reported per thousand localised words. Measures are very useful especially for the project they are generated from, but information in the form of metrics is even more useful, allowing comparison across several projects.

Initial early discussion workshops would help overcome the difficulty with the definitions and requirements, and hence would speed up the implementation process. To date, the main effort has been on data collection, with metrics reporting and analysis of what they mean a bit further behind.

8.A.9: Measure current process and products

This step went very well, and heightened everyone's awareness of the main issues involved in improving the localisation process. I remained responsible for the bug analysis and the defect ratio metrics, and left the productivity and rework measures to the project teams to complete.

The Quality measures are an excellent source of information - they are very important measures for process improvement activities. The two measures used for measuring the current process/products, defect density, and defect analysis, were explained in section 6.9. The positive effects of this exercise were numerous:

- Rather than saying 'there were a lot of type x bugs', the actual quantities, and relative percentage could be quoted
- Areas most in need of improvement were made visible
- US Test Teams were asked to do a similar exercise, by the Exec VP for Product Development

The only negative effect that was noticed was that some people were afraid that the figures would show them up. This perception was somewhat overcome, by stressing that the figures were for their own use, and that they themselves own these findings, and were responsible for suggesting and implementing improved methods on their projects. Some of the results were actually very embarrassing, such as the number of errors introduced due to carelessness, so the fear that there might be repercussions was real. Senior Management in the parent company, who were presented with the summary figures know better than to shoot the messenger, and encouraged improvement in all aspects of the process. This had a very positive effect.

Because this step received so much visibility, certain areas for improvement had to be immediately identified, and quantifiable targets specified. The next version of each project is measured in the same way, in order to provide a true comparison, and to quantify improvements.

Another general outcome of this step, to measure current process and products, is that metrics which are difficult to calculate will be identified, which allows time to either change the metric definition, to change the way some data elements are reported or to write a tool that helps collect the data or to calculate the required metric.

8.A.10: Set improvement targets

Most improvement targets were set as a result of the 'measure current process and products' step. Some of the results were somewhat surprising, so some very specific targets were set for each project as well as some more general targets for the company as a whole. An example of a specific improvement target is to have zero duplicate hotkeys in software handed off to testing. There are also some good examples of generic goals that each Manager has, which have been further defined, and given to the Leads, and further defined again, and given to the Localisers/ Editors/ Testers as a specific goal.

Section 6.10 explains that the metrics recorded on a project are analysed in detail to help plan the next version of the project, and to pinpoint areas where improvements can be made. The example given is how testing effort could be reduced by 50% from one major release to the next.

Section 3.3.2, in discussing programmer productivity, states that sometimes people will work for the figures rather than the quality of the product. It is important to have a variety of measurement criteria in order to get the full picture. An example is that in order to improve the quality of the localised software, the Testers will help the Localisers to test the product, before it is officially handed off for beta testing. While this is an excellent idea, the bugs found should still be counted, if the Testers are spending their effort on it, (although reported in less detail), as the process is still 'do now, fix later' rather than what is known as 'do it right first time'. Taken to an extreme, the official handoff could be delayed until all the localisation testing has been completed, ie, a week before release and hence have no bugs reported. However, when the measure of the number of bugs reported is coupled with the bugs per manmonth (manday) of testing effort, a better picture is obtained. Therefore, to optimise one measure (ie decrease bug density), will show a bad result for another one (ie small number of bugs per manmonth).

8.A.11: Automate the system

Section 6.8 explains the current automated system, which is implemented in three different parts. It includes reporting bug analysis information, effort tracking, and schedule tracking. The implementation of the metrics system has taken much longer than originally anticipated, with the related problem of some new projects waiting for an automated system, before collecting the necessary data (i.e. they are collecting the minimum set of metrics, as defined in section 7.3). Different

departments have implemented the measures in slightly different ways (ie personhours vs. persondays), and some have gone as far as automating the production of monthly reports, which is an area not yet fully defined for the company as a whole.

The bug analysis portion is fine as an initial reporting module, but could be enhanced by including the other defect metrics, and printing out reports with any combination of the metrics from several projects.

The effort tracking system in its current implementation is a great system, but the information is a bit too detailed. The initial set of measures was too granular (see section 8.A.6) in terms of the number of effort categories. Another issue with the system is that it counts hours spent on each task, rather than being less granular, such as number of days, which would suffice. The latest suggested effort categories are listed in Appendix G. The previous granularity of the data collected will be useful for the team itself for very detailed analysis of where time was spent, and the reports that are produced can give very specific information, such as how long was spent on testing hotkeys, and more general information, such as the total time spent localising the product into each language.

Looking at some reports to date, a lot of time is reported in the 'other' category or under 'non-project-specific work'. If one of these categories is present, people will be tempted to report their time under it as it requires less effort than deciding which category to place their days under, if not immediately obvious. Categories such as these therefore should not exist. One department, which is still working with the original data sheets, agreed with my suggestion and has recently deleted the 'other' category, in order to get more accurate effort and cost information.

The effort data on its own is not all that useful for planning and improvement purposes. When effort data is used to calculate metrics such as productivity rates for each function, across all projects, its uses become apparent. An effort tracking system can also be used to measure the cost of the project, by function or by task. Hence, a separate metrics category of 'cost' metrics is not required.

The schedule module, explained in section 7.1.2 c), was developed by a different process team, made up of Program Managers. It's a very good system, with a standard template, which can give current progress information against the planned schedule. Progress curve charts could also be generated from this information.

8.B - Specific Measures

a) Size

Using number of words as the basis of size is a good idea. This allows fairly accurate quality and productivity comparisons across projects. The accuracy of these metrics can be further improved upon, by adding other factors such as the number of menus and the number of dialog boxes to the word count, as menus and dialog boxes require more work to localise than words within an error message, but this can wait for a later implementation of metrics.

The same size measures are used for cost estimates, productivity rates, and defect densities, which makes everything easier to equate and understand. If the cost system requires a change to the definition of product size, then the metrics definitions should follow suit. So far, the use of localisable words as the measure of size is working well, particularly for documentation and help size estimates.

b) Productivity

As predicted, productivity measures during the project received some resistance. Some fair criticism was that the number of dialog boxes, pages formatted, number of words, etc. completed in the month, on a per month basis would take too long to calculate for each person. One department tried to overcome this problem, however, with the reasoning that the total number of words/dialogs/pages in the project is known before the team starts, and one can calculate the amount of work completed each month without too much difficulty, as follows. At the end of each month, how much work remains to be done in the project should be known, therefore by subtracting the amount to be done from the total, what has been completed to date is calculated. If a figure for the amount of work completed in any given month is needed, subtract the previous month's total-to-date from the required month's total-to-date.

For an initial metrics system, end-of-project productivity rates will suffice. All that is required is total effort spent per area, and the total quantity of work items involved in each area. It is then a simple matter of dividing the quantity by the effort, and expressing the number as a quantity per manmonth. This approach is the one that has been adopted by most departments, and is giving good results.

The main benefits experienced are better project planning & control, and the ability to quantify the benefits of new tools in terms of increased productivity.

c) Rework

Documentation is where this rework problem is most noticeable and can cause scheduling problems. Counting the number of pages that contain the errors is coupled with the number of errors found. The main cause of rework is updates from the US, which Ireland currently has no control over. Keeping note of the quantity of updates in the US documentation has prompted managers to look at the content of these changes, before deciding to implement them. If they are purely cosmetic or style changes, then they are ignored, whereas functional changes, such as explanations of how the product works, must be implemented. On one product, it was decided to retranslate the entire documentation, rather than to implement the 1600 changes received in one update. The aim is to decrease the amount of rework involved in each project, expressed as a % of total quantity and total effort. To date, using the combination of quantity and effort provides a great objective measure of the cost of rework - due to both internal and external causes.

The focus of this metric has been amended since what to be included was first defined. External translation vendors are now responsible for formatting the documentation, so that on receipt, a QA check of it is completed, and the vendor is rated according to the quantity of errors found. (see QA rating criteria and definitions, Appendix E).

When the quantity of rework from the Vendors was first counted, the granularity of the defect types was too high, as people were counting every incidence of very minor defects, such as having two spaces between words, instead of one space. This was taking a lot of time to check and report. Having to quantify and report on the quantity of errors found, and the time spent checking the documentation, and fixing the errors found, made it easy to spot that too much time was being spent on this activity.

In order to reduce the time spent checking and reworking the documentation, the categories were changed to make them less granular. The idea of severity of documentation error was also included - ie a deleted screendump is serious, whereas an extra space between words is trivial. The current categories against which the documentation is checked are explained in the Vendor Quality Criteria and Definitions document in Appendix E.

For an initial metrics system, rework measures are very important, and ideally should not be left till the end of the project for reporting them. The rework quantity and effort should be reported on a quarterly (or even monthly) basis for each project, so that timely corrective action can be taken.

d) Effort

Effort measures are used to calculate other metrics, such as productivity and some quality measures, they are also used to calculate internal project costs, and therefore form the hub of any system. The implementation in Microsoft Ireland has recently become a time-tracking system for some departments, which traditionally is not part of the Microsoft culture - some of the fields, such as time spent at meetings, and the 'other' field, should be totally removed. This system is described in some detail in section 7.1.2 a) and its usefulness is analysed in section 8.A.11.

For an initial metrics system, I recommend keeping track on a monthly basis of each team's effort, in terms of mandays spent on the highest-level project areas. Splitting the tasks into about four or five categories per function should suffice, as any more detailed than this, and it is back to counting the number of hours spent on each minor task. A suggested set of categories is included in Appendix G for a localisation environment. The effort tracking system should be based on the steps identified in the process mapping stage (assuming it shows a fairly high-level view of the process).

For future phases of implementation, the system should be fully automated, able to give both a cumulative and non-cumulative analysis of effort, and provide the required reports.

e) Schedule

In the employee process surveys, the fact that the schedules were not perceived as realistic was high on the list of areas that required attention. The main problem here was keeping track of % complete information as the projects progress, at a meaningful level of detail.

The top-level schedule metric, the delta, as described in section 7.1.1, is a great measure, which was originally defined as the difference in elapsed days between the US shipdate and the localised version shipdate. More recently, the delta was redefined to mean the difference between the US and localised release dates. This

metric is one that has been implemented for years, and which has worked really well. If the process is successfully improved, more products should be released within a shorter timeframe (ie. with smaller deltas). This is the main goal, hence some more detailed measurement of the schedule is required, so that projects can be monitored and controlled effectively, from the start.

On the more detailed scheduling measures, the system described in section 7.1.2(c) provides a high-level view of progress against the plan, as shown in section 4.2.1. Producing such a Gantt chart on a monthly basis, and making the schedule available to all project team members, on a read-only basis makes progress visible, and corrective action can be taken early.

A further enhancement would be to produce progress curve charts, and put them on view on the walls where the team-members are situated, and update the chart for each project each month. Patterns can be easily recognised, with good and bad trends being easy to see, and any corrective actions undertaken where necessary.

For an initial metrics system, the minimum schedule metric required is percentage progress against the plan, which can be shown as a Gantt chart. Future implementations would involve further granularity within the schedule, and using the schedule information in conjunction with the effort measures to get a good picture of the current status of the project. eg. progress may be on target, but effort may be 20% greater than planned. The Cumulative Work Packages Complete metric (section 4.2.1(a)) and the Cumulative Effort Distribution metric (section 4.2.2) are ideal for this purpose.

f) Quality

The bulk of effort was spent on defining and implementing quality metrics, and as shown, this is the most important category to point to areas/processes requiring improvement. The software quality measures are explained in section 7.1.3 (b), and the quality measures for Vendors are defined in Appendix E.

The number of active bugs and the total number of bugs at the end of each month are static numbers, which have been reported per project each month. Each of these measures on its own does not provide very much information, but the number of active bugs as a percentage of the total bugs indicates how stable the product is. This is very easy to implement, and team leads keep track of the bugs on all projects.

The bug find rate, ie bug profile by week, is a metric introduced on a pilot project, to see how useful it would be. It is useful to show how stable the product is as testing progresses, and the release date comes closer. It also gives a visual breakdown of each bug type, over each week of the project duration. It is useful both during the project and after the project has released, ie as a post-mortem measure. Both a monthly and a weekly bug profile were tried. In the monthly profile, the data was not accurate enough to give a good picture of whether or not the team were near to being able to release the product. A useful addition to this metric would be to graph active bugs against the total bugs found per week.

Bugs per manmonth (manday) is another metric introduced on a pilot project. It is a useful measure of testing efficiency. Section 8.A.10 explains that the number of bugs reported may be reduced, by involving Testers in performing a full localisation pass on the product, with the Localisers, before handing it off to beta testing. This was done for the help files on one language of the pilot project, which gave a number of 5.3 bugs per manmonth on the help files for that language, against 27 bugs per manmonth on the other language, which had not been 'pre-tested' by the Testers quite as much. Looking at these figures, one would say that 5.3 help bugs per manmonth indicates a lot of time wasted testing the project, as the total number of help bugs could have been found in less than a manweek. This number should increase as the process improves. As an example, automating the testing for a common bug type should ensure that the bugs are found early, and take less time to find (since the tool can be run overnight). Hence, the number of bugs found per manmonth of testing is useful to demonstrate the effect of increasing the amount of automated testing.

The best implementation of the bugs per manmonth metric is to report it at the end of each project, and then use this metric to aim for a higher number of bugs per manmonth on the next version of the product.

Categorisation of bug origin is a simple measure, which can be produced monthly or at the end of the project. This has been implemented for all projects. A product with a high number of bugs requiring a code fix indicates that firstly these issues should be communicated to the US Development team so they can produce global code, and also time should be spent testing for international functionality issues, whereas on a product with none or very few of these errors, an automated

functionality acceptance test run against each build would suffice, along with localisation-specific testing.

Bug Density is based on the well-known Bugs/KLOC metric, and has been calculated retrospectively for the previous version of each of the major products. It makes it possible to compare quality of localisation across products of different size and type. Again, it is a very simple measure, ideally reported at the end of project to start with. On recent products, there has been a decrease in the bug density as compared with the previous version.

Bug severity is another end-of-project metric, which is useful for getting an overview of the bugs found, and is similar in use to the bug origin measure. It has been implemented on several projects, in conjunction with the bug origin measure, to get a more detailed view.

Defect analysis, ie types of localisation error is probably the most useful quality measure that has been implemented. As explained in section 8.A.9, this information was obtained and charted as pareto charts for the previous version of all major projects (over 50 databases in total were queried). This went extremely well, and provided details of areas that needed a lot of improvement, such as text errors, eg small oversights such as mis-spellings, untranslated text and duplicate hotkeys. This defect analysis activity led to some causal analysis activities, which in turn has led to some new tools being developed and new methods introduced, to reduce/eliminate some bug types.

The following table gives a summary of the main benefits of each metrics category introduced on the projects.

Measurement Category	*Main Benefits*
Size/Complexity	Used as basis for calculating other measures Allows normalisation of measures to enable cross-project comparisons Helps Project Planning
Productivity	Helps Project Planning and Control Measures effects of new processes/tools
Rework	Identifies Cost of Non-Conformance Identifies areas for improvement
Effort	Helps Project Planning and Control Helps track Project Cost
Schedule	Helps Project Planning and Control Determines Project Progress
Quality	Helps determine when to stop testing Identifies areas for improvement Measures effects of new processes/tools

9. CONCLUSIONS and FUTURE WORK

In order to first control the process, then to improve it, a measurement system should be introduced. The measures chosen should help point to areas that require improvement; improvement goals should be set using these measures; new improved methods should be introduced, and then their effects quantified using the same set of measures. For an initial metrics implementation, global, end-of-project measures should be introduced. As the process becomes under control and matures, more detailed measures should be implemented, in phases.

Metrics systems should be kept simple to start with, and then expanded and built upon when the need arises, ie when the implemented measures on their own are not sufficient to measure the changed process, or when the process will support more detailed measurements.

The initial 3 months of implementation should be as a pilot project. A project of average size and complexity should be chosen, with the project team reporting the suggested metrics for the three-month period, along with issues and problems encountered in collating and reporting the data. Process improvement is a separate activity to metrics implementation, so the 3-month trial should concentrate on determining if the chosen measures:

- Enable effective Planning
- Enable control of the process
- Determine project progress

If the measures firstly enable control of the process, then the same measures can be used to quantify process improvement efforts. Ideally, quality management activities should be implemented alongside the metrics, to ensure full benefit from each of these processes, as follows:

- Enable benefits of new tools/methods to be stated in quantifiable terms
- Identify areas in need of improvement
- Enable improvement goals to be set in quantifiable terms
- Demonstrate improvement

9.1. Implementation Time

In order to implement a metrics system which will be a success, all the process steps described must be implemented in full. It will take time to implement these, and there is no quick fix available. Failing to implement any of the steps in full can lead to some problems further on. For example, training only half of the people leads to lack of understanding of the quantity and granularity of measures required for successful implementation, which leads to slower implementation of the metrics system.

The time required to implement a metrics system, which is customised to the methods and processes of each company, will depend on the size of the company, and the current availability of required data. A company should allow at least 6 to 9 months for going through the 12-step process, and introducing the first simple measures. Hofstadter's law is an appropriate quote here:

It always takes longer than you think, even taking Hofstadter's law into account

The metrics implementation process should be managed in the same way as a normal development project, and has similar schedule compression constraints. The main reason that it cannot be compressed into, say, 1 month, is that it takes time for all employees to internalise the need for measurement, and its benefits in terms of improving the process. However, if a quality culture is already firmly in place, metrics implementation will be easier than if it is all a new endeavour, or if the quality culture comes later.

9.2. Process

The 12-step process defined will work for a variety of development environments. The metrics described can be adapted for use in a localisation environment, as demonstrated, and a similar adaptation could also be achieved for a software maintenance environment, or any other variation of the process. There are a few other metrics implementation models, but they tend to be either distinctly top-down (based on the organisational goals only) or distinctly bottom-up (based on the current process only), which both have some inherent flaws. The 12-step process presented is an optimal approach, as it addresses the issue from both ends.

Books, seminars and conferences often concentrate on explaining the need for metrics, and briefly describe a few measures, but my experience has been that a transfer of learning does not take place, ie the readers/attendees leave without knowing how to implement these measures in their own company. At a recent conference, the attendees who attended the 'implementing metrics' tutorial were very disappointed with it, as it covered things they already knew and had internalised, such as why metrics are useful, followed by a long list of popular measures. What these people wanted was a process they could adapt to their environment, in order to implement a useful metrics system, with a full explanation of how the measures are used. This dissertation successfully addresses this need.

It is useful to know what the metrics are, but how to implement them, and what information they give with respect to the development process is even more useful to seminar attendees. There is a Chinese proverb, which is appropriate to metrics implementation:

> *Give a man a fish, and you feed him for a day;*
> *Teach him how to fish, and you feed him for life.*

The key to successful metrics implementation is to keep it simple, ie measure neither too much nor too little, and understand why each measure/metric is being used. The right amount is determined primarily by the process map, the current capability maturity level, and the corporate improvement goals. Focus on the needs, and the implementation will be successful.

The metrics system should be designed by a group of people, who in turn solicit feedback from and provide progress reports to a group of other employees, so that each of the company employees is kept informed. Involve as many people as possible in the metrics definition process, so that full understanding of the issues is obtained as early as possible, which gives the metrics system a good chance of survival. This group should define what will be measured, with the final decision on granularity, and collection responsibility being the decision of the managers in each Department.

9.3. Metrics

The Metrics definitions sections (chapters 4 and 5), concentrate on describing hard measures, eg, based on lines of code, time spent, and bugs reported, ie those that can be extracted from data that is physically available. One of the early process steps (Section 3.3) involves soft measures, eg the employee and customer surveys, ie those that are based on attitude or perceptions concerning the process or product. Both hard and soft measures are useful and effective in getting a full picture of the process, from planning through to implementation.

Soft measures can be hard to implement, as the emphasis is on individual expectations and requirements. The surveys are very useful for evaluating current perceptions, and providing/receiving feedback.

A combination of measures is required for the metrics system to effectively point to areas where process improvement is required, and to quantify the improvement, when the process is changed. Measuring and optimising only one aspect of the process is similar to squeezing one part of a balloon - the other parts will pop out. For example, if defect density is used to measure the quality of the product, and effort is not measured, the defect density might decrease by 50%, but with an increased effort of 70%. If effort is not also measured, there will be a false sense of security.

For a company new to metrics implementation, and looking for the optimum metrics to start with, to enable basic evaluation of the current processes, leading to some initial process improvement work, I would suggest basic effort, schedule and productivity measures, with some detailed quality and rework measures. Some level-specific metrics are suggested in section 3.4.2.

9.4. Metrics w.r.t. Improvement

Many companies have a lot of the required data available in some format, but it is not used as a measure for process improvement. For example, many companies record time spent on various project activities for cost accounting purposes, which can be adapted and used for both effort and schedule metrics. The same can be said for bug reports - most companies have some form of written bug reporting method, which could form the basis for the quality metrics.

A Quality culture provides a positive environment for measurement and process improvement. A metrics system, without the infrastructure of a quality/process improvement culture may fail within a short time of implementation, or at least take much longer to implement. A quality culture involves rewarding, rather than shooting the messenger. People must feel free to be honest about all the measures, and be encouraged to bring issues into the open as early in the process as possible.

Another important issue is the feedback loop. Give feedback to the people providing the data about how it's being used, the benefits of the measures collected, and the results of using these measures. Reporting data every month or at the end of the project, that is never heard of again, is not conducive to a process improvement culture/environment.

The most beneficial way of implementing the measures for process improvement is to measure current processes and products first, as described in sections 3.9 and 6.9. This provides excellent baseline data, from where improvement targets can be quantified, then measured as time progresses. This step provides the historical data, by measuring past projects retrospectively, using the defined measures.

The improvement targets obtained from looking at the worst results of past and current metrics are the best ones to start with, as this helps focus the 20% of the improvement effort to give 80% of the results. Start with something easy to show some improvement in, demonstrate this improvement, and build on the resultant success.

By following the process described, and implementing a subset of the metrics that have been explained, a successful metrics system to facilitate improvement will be implemented.

9.5. Future work

As the metrics for my case study were being implemented, the focus of the localisation effort was changing. In some instances, the focus changed almost overnight, and resulted in a re-organisation of one of the Departments. The focus here has changed to external vendors completing all of the localisation, including software localisation, so some measures needed to change and now have a different focus for that department. Over the next two years, other departments will follow suit. I have begun to address this need, by devising a Vendor QA system, containing both hard (no. errors found, amount of rework) and soft (two-way feedback on project management processes) measures. An example of the feedback form the vendor sends back is in Appendix C, and an example of the project follow-up form sent to the Vendor is in Appendix D. Rework and quality are the most important measures of the vendors, as described in Appendix E. Future work in the vendor area includes a Vendor certification/approval system, based on the clearly defined measurement criteria, and process capability assessment, based on the Capability Maturity Model.

From the viewpoint of my own work in Microsoft Ireland, I have started to work further on the format of how, when and to whom each metric will be reported. The main area of further work is to enable the seamless comparison of measures across projects in different departments. For this to happen, I need to firstly devise a standard project post-mortem template (see draft template, Appendix I), and secondly to design and implement a database system containing the metrics calculated for each project, from which desired reports could be generated whenever required. Different levels of information would be available, so that summary information can be obtained, and further levels of detail would then be available if more specific information is desired.

The bug analysis portion of the work on measuring current processes and products has been adopted by several Test Teams in the US parent company. An additional step would be to encourage them to introduce some more of the suggested measures, particularly the quality measures, using the implementation process as described here.

Appendix A

Glossary of Terms

Benchmarking	A process whereby specific quantifiable standards are set in terms of metrics, which all products are compared against.
Bug	A difference between the way the product behaves and the expected behaviour as described in the product specification
CBT	Computer Based Training program, ie Tutorial system.
Defect	I use the term defect as a synonym for bug
Deliverables	Completed work items which are returned by an external company, ie Vendor to the contact person in our company
Dialog box	A graphical box containing a variety of options to choose which appears when a menu option is selected in a product with a Graphical User Interface
Executable	A compiled file containing programs, screens and menus which can be executed. Executable files have a .EXE file extension
Handoffs	The name given to deliverables which originate internally
Headcount	Required staff numbers
KLOC	Thousand Lines of Code
Localisation	The process of translating all parts of a product (software, documentation, help, templates, etc.) into a particular language and making sure that all examples and currency/date formats used are appropriate for the specified market
Localisability	The ease with which a product can be localised, without introducing functional bugs
Localiser	An employee responsible for localising a product into a target language
Measure	A standard or unit of measurement, ie dimension of something
Metric	I generally use the term metric as a synonym for measure

Milestone	A point in time for which a specific project goal has been set, in terms of quantity completed, etc.
NLS	National Language Support. Date/time/number formats and sorting order specific to each language
Pareto Chart	A method of charting information which displays categories in decreasing order, and as a percentage of the total. Demonstrates that 80% of the errors are due to 20% of the causes
Popup	In a helpfile, by selecting a highlighted word/phrase within a topic, a small box appears on the screen which explains the word or phrase
Priority	The level of importance of a defect, in terms of how quickly it needs to be fixed
PU	Product Unit, a department responsible for a specific category of localisation products
PUM	Product Unit Manager
Release date	The date when the software is signed off to our manufacturing facility for subsequent duplication and shipment
RTS	Resource Tracking System. Internally developed effort tracking database
Screendump	A bitmap picture of a screen showing part of the user interface of a product
Severity	The level of impact a defect has on the use of the system
Shipdate	The date the product leaves the manufacturing facility after being duplicated and shrink-wrapped
Subsidiaries	Companies around the world which market and sell our localised products
Team Lead	A first-line Manager, responsible for the activities of his/her team on a specific product
Work Package	Detailed small-job, many of which form the total work for the project. They tend to be of roughly-equal effort to complete
Vendor	A company contracted to localise software or translate documentation or help files for a specific product and language

Appendix B

SUBSIDIARY PRODUCT EVALUATION FORM

PRODUCT NAME & VERSION: SUBSIDIARY: RELEASE DATE OF PRODUCT:	LANGUAGE: SUBSIDIARY PM:
Please fill out the evaluations below for both PRODUCT and PROCESS, and return to: WPGI PRODUCT UNIT MANAGER:	

PRODUCT Evaluation Summary

Please rate the quality of the product below by area as follows:

5 = EXCELLENT
4 = GOOD
3 = SATISFACTORY
2 = IMPROVEMENT NEEDED
1 = UNACCEPTABLE

Evaluation per Product Area	Quality Rating
Software	
Packaging	
Help	
Printed Documentation	
CBT	
Timeliness	

OVERALL PRODUCT QUALITY RATING	

PRODUCT Evaluation Detail

Please rate the statements below as follows:

4 = STRONGLY AGREE with the statement
3 = AGREE
2 = DISAGREE
1 = STRONGLY DISAGREE

SOFTWARE

Statement for Product Area	Exe	Add-ins	Setup
The localization is stylistically correct			
The approved terminology was used			
The page/screen layout matches the English product			
The localization is functionally correct			
Examples and templates have been appropriately localised			
Overall, the localization is of good quality			
Other comments			

USER EDUCATION

Statement for Product Area	Doc	CBT	Help	Packaging
The localization is grammatically correct				
The approved terminology was used				
The page/screen layout visually matches the English product				
The localization is functionally correct				
The product terminology is correct				
Examples and templates have been appropriately localised				
The indexes are comprehensive and accurate		*N/A*		*N/A*
Overall, the localization is of good quality and meets market requirements				
Other comments				

OVERALL PRODUCT COMMENTS

With which 2 or 3 items were you particularly pleased?	**COMMENT:**
Which 2 or 3 items caused the most number of PSS calls?	**COMMENT:**

PROCESS evaluation

Statement	Rating	Comments
The glossaries were received on time, to enable feedback to be given		
The design specs. were received on time, to enable feedback to be given		
The localization specs. were received on time, to enable feedback to be given		
We were kept up-to-date with schedules on a fortnightly basis		
The agreed delta was met		
Our queries were answered in a timely manner		
Our input was taken into consideration while localizing the product		
Beta copies of the software received as agreed		
Feedback on beta version was acted upon		
Feedback from PSS has been taken into account		
We were invited to participate in the Black Team testing		
We were kept informed of progress, problems and solutions throughout the project		
Overall, the service we received throughout the localization of this product met our requirements		

OVERALL PROCESS COMMENTS

With which 2 or 3 items, relating to process, were you particularly pleased?	COMMENT:
Which items, if any, caused you difficulty?	COMMENT:

Appendix C

Vendor – Microsoft Feedback Report

COMPLETED BY:	DATE:
PRODUCT NAME & VERSION:	LANGUAGE:
VENDOR:	VENDOR PM:
RELEASE DATE OF PRODUCT:	WPGI PM:

GENERAL

Please tick the appropriate numbered reference below.

Glossaries:

[ ] 3 Software and General Glossary received during Vendor preparations phase; approved prior to Vendor localization by Language Services and Subsidiaries; updated systematically throughout the project, enabling costs to be maintained and acceptable quality achieved on schedule.

[ ] 2 Software and General Glossary received prior to Vendor localization; approved by Language Services and subsidiary, not finalized; but updated systematically throughout project, costs increased up to 10% of original cost; acceptable quality achieved on schedule.

[ ] 1 Software and General Glossary not received until after Vendor localization started; not approved by Subsidiary or Language Services, causing poor quality; delays, rework and increased cost of Vendor localization greater than 10% of original cost.

Comments Vendor Project Manager:

International Specifications:

[ ] 3 International Specifications received prior to Vendor localization; updated systematically throughout the project, in anticipation of Vendor localization issues; enabling costs to be maintained and acceptable quality achieved on schedule.

[ ] 2 International Specifications received prior to localization; not finalized; but updated throughout project in, but requiring Vendor inquiry of Specification issues; costs increased up to 10% of original cost; acceptable quality achieved on schedule.

[ ] 1 International Specifications not received until after localization started; not updated, requiring constant Vendor inquiry, delays in Vendor localization; rework; and increased cost of Vendor localization greater than 10% of original cost.

Comments Vendor Project Manager:

Deliverables:

[ ] 3 Preliminary delivery dates and Localization Kit [1] planned during Vendor preparations phase, deliveries achieved according to all agreed dates, milestones and costs, facilitating project management[2] in the Vendor and quality standards to be maintained

[ ] 2 Deliveries re-scheduled during Vendor localization, agreed with Vendor, causing project management issues in the Vendor to arise in order to maintain quality standards; and costs to increase up to 10% of original costs.

[ ] 1 Deliveries not communicated clearly to Vendor or agreed in advance, deliveries late, causing serious project management issues in Vendor to arise, causing costs to increase greater than 10% of original costs, and compromising quality to achieve major project dates and milestones.

Comments Vendor Project Manager:

[1] Tools, source files, beta software.

[2] Resource planning, scheduling of workloads, quality maintenance, cost control, and delivery targets.

Training/Instructions:

[ ] 3 Training needs identified. Training and Instructions were planned in advance of localization; needs were identified to meet Localization goals, timely, supported during Localization, evaluation acted upon.

[ ] 2 Training needs identified. Training and Instructions received after Localization start, useful to meeting Localization goals but not directly relevant to project specifics, further instructions required during Localization causing non-critical delays and rework.

[ ] 1 Training and Instructions not received or needs identified, instructions not provided, rework and delays to major milestones caused.

Comments Vendor Project Manager:

Feedback:

[ ] 3 Quality Assurance checks carried out throughout localization, concise actionable feedback supplied, Vendor feedback evaluated and acted upon,.

[ ] 2 Quality Assurance checks carried out through localization, feedback unclear, requiring further clarification and rework, no clear response to Vendor feedback requiring re-work, but not critical to project.

[ ] 1 No Quality Assurance checks carried out after Main QA, no further feedback, Vendor feedback not responded to, causing project rework and slips in major milestones.

Comments Vendor Project Manager:

Communications:

[ ] 3 Weekly status report and schedule update from PM, informing of upcoming deliveries, progress and forthcoming issues needing Vendor attention; frequent contact with Vendor teams during Localization, ensuring progress and redressing critical issues within 24 hours.

[ ] 2 Irregular status report and schedule updates, intermittent contact with Vendor teams, some delays in communicating information, but not causing Vendor project management issues.

[ ] 1 No status reports and schedule updates, poor contact with Vendor teams, causing project management problems in Vendor and rework. Queries not responded to causing rework and major milestones to slip.

Comments Vendor Project Manager:

WHAT WENT WELL?	
Which 2 or 3 items do you think went particularly well?	**COMMENT:**

WHAT SHOULD BE IMPROVED?	
Which items caused you most difficulty and should be changed for the next project?	**COMMENT:**

Signed: Date: 21 September, 1993

Appendix D

PROJECT FOLLOW UP

Project Management, Formatting, Functionality, Language

PRODUCT NAME:	VERSION:
LANGUAGE	
PROGRAM MANAGER:	
ORIGINAL RTM:	ACTUAL RTM:
DID PERFORMANCE MEET OVERALL BUSINESS OBJECTIVES?:	
SIGNED:	DATE:

PROJECT MANAGEMENT

Please tick the appropriate numbered reference below that most fits the description of how the project was handled by the Vendor

General Project management:

[ ] 3 Understand and comply with agreed instructions, training and process; seek support and information where necessary; achieve all major project dates and milestones, remain flexible to change; adhere to all quality standards maintain cost control, add value to MS first time

[ ] 2 Understand and comply with agreed instructions, training and process; require extra support and information to meet quality standards and project dates and milestones , negotiate over changes and costs.

[ ] 1 Do not follow process, training an instructions, fail to make project dates and milestones and quality standards, do not seek support and information where necessary, cannot accept change, costly to MS, add no value to MS.

Comments MS Program Manager:

Timeliness:

[ ] 3 Deliver according to all agreed major project dates and milestones; maintaining quality standards and costs.

[ ] 2 Deliver according to newly agreed dates during localization, adjustment required to dates in WPGI but not affecting major dates and milestones; maintaining quality standards and costs.

[ ] 1 Do not deliver on time, do not inform WPGI of pending slip, cause project release/ship date to slip.

Comments MS Project Manager:

Communication:

[ ] 3 Send weekly status report to PM, accept and provide feedback, maintain two-way contact with teams during localization, seek information and support where necessary, identify problems and communicate issues to WPGI, facilitating delivery of acceptable quality product.

[ ] 2 Sometimes send weekly status report, intermittent contact with teams during localization, feedback not always acted upon, do not seek information and/or implement changes until after QA results received in order to maintain quality standards.

[ ] 1 Send no reports, maintain poor contact with teams, do not act on any feedback causing rework, recurrence of problems and unacceptable quality.

Comments MS Project Manager:

VENDOR EVALUATION SUMMARY

DOCUMENTATION

Category	Overall Rating
Language	
Formatting	
Functionality	

HELP

Category	Overall Rating
Language	
Functionality/formatting	

CBT/CUECARDS/WIZARDS

Category	Overall Rating
Language	
Functionality/Formatting	

SOFTWARE

Category	Overall Rating
Language	
Functionality/Formatting	

Appendix E
Vendor Quality Criteria and Definitions

Language Criteria

Accuracy of Translation/Localisation.
Does the translated/localised text factually cover the English source text, has all necessary text been translated/localised, does the software contain spelling errors, typing errors?

Terminology.
Does the translation/localisation adhere to the glossary, have non-glossary terms been translated appropriately and is the terminology consistent within the product family?

Style.
Is the translated/localised text clear and concise, is the tone direct, modern and friendly, have all examples been localised appropriately?

Language.
Are grammar, spelling and punctuation correct?

Country.
Have all country-related (NLS) data been implemented correctly?

Index. (Documentation only)
Has the index been translated/adapted correctly?

Help/CBT/ Cuecards/Wizards Formatting and Functional Criteria

Topic Errors
Are there any broken links for jumps or pop-ups? (Use YETI to check).

Footnote Errors
Is there any missing footnote information, or formatting problems in the footnotes? (Use FOOF to check).

Graphics Errors
Are the art pieces displayed at the proper places, and are all pieces included? (Use HlpDrv. to step through); are the art pieces completely visible, completely translated, and displayed correctly on the required display units? (compile a Help file which only contains the art, and view this help file with the winhelp engine).

Layout Errors
Does the layout obscure the meaning of the text? (Use HlpDrv. to step through); Is the layout affected by resizing the window of the Help file? (Use HlpDrv. to step through, and resize the screen to approx. 40% of its normal size).

Documentation Formatting and Functional Criteria

Font Errors
Have the correct standard fonts been used throughout the documentation? Are type sizes all correct?

Template Errors
Have standard templates been used, and not changed? Are page margins correct?

Format Errors
Has overtyping been performed correctly? Are words and paragraphs formatted and spaced according to standard practice?

Page Format Errors
Is page numbering correct? Are the page-breaks where they should be? Are all page headers correct?

Art Errors
Are art references correct, and art pieces in the appropriate positions?

Index errors
Are all index codes present? Are all paragraph tags present?

Output Errors
Are postscript files naming conventions followed and are the files in the correct file format? Is the runlist information correct?

Inconsistencies
Is the documentation consistent with the software, Help, CBT, Cuecards, and Wizards with respect to hotkeys, status bar, alert messages, etc.?

Graphic Errors
Are all art pieces correctly localised, appear in the appropriate places, with accurate callout references?

Localisation Errors
Is the localisation technically accurate, showing a thorough understanding of the product by the vendor?

Software Formatting and Functional Criteria

Text Errors
Is required text present and displayed correctly, with correct formatting and alignment, not truncated and no garbage, 'funny-face', characters present?

Dlg. Box Errors
Are there any misalignment, missizing or mispositioning errors concerning either dialog boxes or their components (check boxes, buttons etc.)?

Hotkey Errors
Are there any missing or duplicate hotkeys?

Macro Errors
Can macros be recorded correctly, and do supplied macros run without error?

Functional Errors
Does the software function according to expectations?

Quality Definitions

Ratings and Rating Parameters

The ratings resulting from the QAs are based on:
10,000 words of **Documentation**
5,000 words of **Help**
5,000 words of **CBT, Cuecards, Wizards**

5 Exceptional Quality

Doc, Help, CBT, Cuecards, Wizards:

Language: Describes functionality very accurately. Info very accessible, consistent terminology, style. No grammar, punctuation, spelling errors. All page refs correct. Complete index. Correct NLS data.

Formatting: All standard fonts and templates used. No paragraph style or character formatting errors. All art referenced correctly. No index or TOC formatting or reference errors. Help is complete with perfect alignment and no display errors.

Functionality: Documentation and help fully consistent with software. Correct art pieces inserted throughout, with all callout references in documentation accurate. Technically accurate localisation, with no misleading statements, showing a thorough understanding of the product. No topic or footnote errors in help.

Software, Setup:

Language: No language bugs. Completely, accurately localised. Examples very well localised. UI terms consistent, adhering to MS guidelines, accurately describing functionality. Fully correct NLS data.

Formatting/Functionality: Dialog boxes all correctly sized. No duplicate hotkeys. All text items localised, aligned correctly and fully visible on screen. Macros correctly localised and fully functional.

Packaging:

Language: Info very accessible. Very well written, consistent, well localised. No language, terminology errors. Fully correct NLS data.

4 Excellent Quality	***Doc, Help, CBT, Cuecards, Wizards:***
Language:	Describes functionality accurately. Most info readily accessible. Consistent terms, style. Most examples well localised. Very few grammar, punctuation, spelling errors. Very few inconsistencies. Very few errors in page refs or index. No critically wrong NLS data.
Formatting:	All standard fonts and templates used. Documentation well-formatted with few style or character formatting errors. Art correctly referenced. Help is complete with good alignment and few minor display or graphics errors.
Functionality:	Documentation and help very consistent with software. Localisations adequate and technically accurate, demonstrating a good understanding of the product. Correct art pieces inserted throughout, with accurate callout references. Help functionally correct with no noticeable topic or footnote errors.
	Software, Setup:
Language:	No language bugs. Completely, accurately localised. Examples well localised. Few UI terms inconsistencies. No critically wrong NLS data.
Formatting/Functionality:	Macros correctly localised and functional. Localisation does not break any aspects of functionality. Software contains no severity 1 or severity 2 localisation bugs. Few minor or trivial errors present.
	Packaging:
Language:	Most info accessible. Well written, consistent, well localised. Very few language errors. No terminology errors. No critically wrong NLS data.
3 Good Quality	***Doc, Help, CBT, Cuecards, Wizards:***
Language:	Describes functionality adequately. Info generally accessible, writing style acceptable; terms generally consistent. Examples localised fairly well. Few grammar, punctuation, spelling errors. Few inconsistencies, errors in page refs or index. Fully correct NLS data.
Formatting:	Adequately formatted for market needs. Contains no major formatting errors which impact on the user's ability to understand the documentation/help/CBT. Few errors in character, paragraph and index formatting.
Functionality:	Functionally correct, meeting market requirements. Few localisation inconsistencies, in non-significant areas.
	Software, Setup:
Language:	No language bugs. Adequately localised. Examples adequate. UI terms adequate. Few errors in NLS data.
Formatting/Functionality:	Software well-localised, with few user interface errors which do not impact on the product's ability to satisfy the user's requirements. Functionality not impaired as a result of localisation.
	Packaging:
Language:	Info generally correct. Writing style acceptable, consistent. Acceptably localised. Few language, terms errors. Few errors in NLS data.

2 Needs Improvement	***Doc, Help, CBT, Cuecards, Wizards:***
Language:	Does not adequately describe functionality/describes some incorrectly. Info difficult to access. Writing style not appropriate, consistent; terms not always consistent. Many examples not well localised. Many grammar, punctuation, spelling errors. Many inconsistencies, errors in page refs or index. Many errors in NLS data.
Formatting:	Some changes made to standard fonts/templates supplied. Many format errors. Needs general improvement to comply with market requirements. Contains obvious errors that should have been caught and corrected before handoff.
Functionality:	Help contains topic and footnote errors, demonstrating that tools provided have not been used properly. Several inconsistencies found between documentation, help and software.
	Software, Setup:
Language:	Language bugs. Inadequately localised. Examples inappropriate. UI terms often inconsistent. Many errors in NLS data.
Formatting/Functionality:	Contains some severity 1 or 2 localisation bugs. The number of minor localisation errors found shows that the software has not been adequately tested before handoff.
	Packaging:
Language:	Info difficult to access. Poor writing style, inconsistent. Not well localised. Many language, terms errors. Many errors in NLS data.
1 Unsatisfactory	***Doc, Help, CBT, Cuecards, Wizards:***
Language:	Product difficult to use (lack of info), no. of discrepancies between s/w and doc. Info difficult to access. Writing style poor; terms inconsistent. Few examples well localised. Numerous grammar, punctuation, spelling errors. Many serious inconsistencies, errors in page refs or index. Unacceptable errors in NLS data.
Formatting:	Difficult to use due to some serious formatting and layout errors. Standard templates/fonts not adhered to. Requires a significant amount of rework to comply with market requirements.
Functionality:	Many inconsistencies between doc, help and software. Several jumps/popups not functioning in help. Poor understanding of product, leading to inadequate technical localisations throughout.
	Software, Setup:
Language:	Seriously inadequate localisation. May present significant usability problems. Inconsistent UI terms. Unacceptable errors in NLS data.
Formatting/Functionality:	Contains a significant number of localisation bugs. It takes longer to report the bugs than to test the product, demonstrating the lack of any QA check before handoff. Significant amount of rework required to bring the software to an acceptable standard.
	Packaging:
Language:	Info not accessible. Writing style very poor, inconsistent. Poorly localised. Very many language, terminology errors. Unacceptable number of errors in NLS data.

Implementing Metrics

What

The Metrics are broken down into the main functional areas of each project. Within each functional area, the main activities have been defined. The data collected is a measure of the project, not of individual performance. The metrics are collected for the use of the project team itself, as will be explained further in this document. Any measures that are not applicable to a particular project should be shaded out in the data sheets by the person in the PU who sets up the files for each project.

Who

Generic metrics sheets are provided for each Unit to adapt to include the names of their projects. Different people are responsible for different areas of the metrics sheets, although the responsibility for the accuracy of the data belongs to everyone on the team.

The metrics can be prepared and analysed for several recipients. First and most importantly for the team working on the project, secondly for the PUM, and thirdly for people outside the PU, ie those in Ireland and in Redmond that are interested in the progress of the localisation process. We need to be able to show all interested parties how we have improved and how well we are doing.

When

In order to keep track of most of the metrics, a weekly sheet should be filled in, which will be rolled up monthly to provide data for monthly reports, and give a good insight as to where the project is currently at. Some of the measures are only required monthly, whilst a few are entered at the end of the project. At the end of the project, the metrics will form a major part of the Post Mortem report.

Where

Each PU will create its own directory structure on one of their servers, and will create files for each week and each month of the project for the specific metrics categories. At the end of the project, all sheets should be available so that a detailed analysis can be carried out and the relevant reports on productivity rates, schedule, rework and quality can be produced.

Why

Our overall company goal is to improve our localisation process in terms of time, quality and cost. In order to do this, we must have some tangible measures of success. The metrics that have been selected are indicators that will help us identify our improvements against the following specified goals:

- Reduce the cost of rework
- Improve the quality of our localised products
- Improve the productivity of our teams
- Improve the accuracy of the estimates of effort and timescales for projects

How

The following pages contain guidelines for filling in the data on each of the metrics sheets, in addition, the following general guidelines should be noted:

1. For data on Personweeks, include all time spent working on the area, including overtime. 8 hours constitutes one day. Round the number of weeks to the nearest week. This is not a time tracking system, so if there are 2 people who were formatting for the month, one of whom was in for the four weeks, with no O/T, and the other person was out sick for 1 of the weeks, then the total to be entered is 7 person weeks.
2. Rework causes are important and should be as accurate as possible, so that we can work to eliminate the major causes.
3. Unapplied time is defined as time spent on areas other than project work, eg when someone is between projects, and waiting for something from the US.
4. Bug analysis will be automated and the standard queries provided on the RAID servers.

Releases

Measurement Item	Entry required	Definition	Who	When	Benefits/trends	Useful calculations for reports
Product Releases	Number of releases	The product is released when it has been fully signed off to Manufacturing	Program Manager	Monthly		
Software Re-releases	No. Software re-releases	A product re-release occurs when a new part number is generated for one or more disks of the product	Program Manager	Monthly	This measure shows us the true quality of our software output. Ideally, this number should decrease and approach zero	no. re-releases divided by no. releases, as a percentage
Film Re-Release	No. of film re-releases	A film re-release occurs when new film is generated for the product, as a result of an error made by WPGI	Program Manager	Monthly	This measure shows us the true quality of our film output. Ideally, this number should decrease and approach zero	no. re-releases divided by no. releases, as a percentage
Disk Releases:	Number of disks released	The number of Golden Master Disks signed off to Mfg - all media.	Program Manager	Monthly		
Disk Re-releases:	No. disks re-released	A disk re-release occurs when only the MDC# changes, not the part number	Program Manager	Monthly	This number should decrease and approach zero	no. re-released disks divided by no. released disks, as a percentage.
Delta:	Current Delta	The delta is the difference between the US release date and the localised product release date, expressed in elapsed days	Program Manager	Monthly	As our processes improve, the deltas should get shorter.	Obtain from Project Status Report

Documentation

Measurement item	Entry required	Definition	Who	When	Benefits	Useful calculations for reports
Effort - Person weeks	No. p/weeks format	No. weeks Editors spent formatting the documentation (include O/T)	PTL	Monthly	Measuring effort spent has two main uses - firstly, it improves planning, costing and scheduling of the project, secondly personweeks is used as the basis for the productivity metrics below	
	No. p/weeks add-ins	No. weeks Editors spent on misc add-ins (if applicable)	PTL			
	No. p/weeks prep	No. weeks Artists spent prepping art	PTL			
	No. p/weeks review	No. weeks Localisers spent reviewing the documentation	LTL			
	No. p/weeks rework	No. weeks Editors, Artists and Localisers spent on rework	PTL & LTL			
	No. p/weeks Unapplied	No. weeks Editors, Artists, Localisers were not working on the project	PTL & LTL			
Pgs. Planned	No. pages	The total originally planned no. pages	PM	Project start		
Pgs. formatted:	No. pages formatted	The number of pages formatted (ie edited/cleaned up, etc.) by the Editor(s)	PTL	Monthly	Measuring throughput will confirm productivity improvements as a result of process or tool improvements. eg how will the MSD affect the no. of pages that can be formatted in a personweek? The cumulative % of the total to be formatted will assist in managing the project.	Pages formatted divided be the personweeks (s) of effort and cumulative pages as a % of total to be formatted
Pgs. reworked:	No. Pages	The total number of pages rework	PTL	Monthly	should decrease and approach zero	Pages reworked, divided by the total planned pages, expressed as a %
Add-ins (if applicable)	No. pages of add-ins	The quantity of add-in items, such as misc. fliers, offers, etc.	PTL	Monthly	Much time and effort on some projects is spent on areas not covered by the formatting and art prep columns, which should be measured in order to see where improvements can be made	
Loc Review:	No. pages reviewed	The number of pages reviewed from translation by the localiser	LTL	Monthly	Ideally, this should decrease and approach zero, if the translators are getting it right first time.	Pages reviewed divided be the no. of pages translated, as a %

Measurement item	Entry required	Definition	Who	When	Benefits	Useful calculations for reports
Pieces planned	No. pieces	The total originally planned no. pieces	PM	Project start		
Actual Art	No. art pieces completed	The number of pieces of art completed (ie shot and prepped).	PTL	Monthly	New art procedures and techniques may impact the efficiency with which art can be completed, hence the need to measure the current throughput.	Actual art divided be the personweeks of effort; and cumulative art as a % of the total planned
Art Re-worked:	No. Pieces	The total number of pieces of art reworked	PTL	Monthly	should decrease and approach zero	
Loc QA:	No. Pages	The number of pages QA'd by the localiser.	LTL	Monthly	If the Quality of the original work increases, the QA checks should go much faster, hence throughput of pages will be higher.	Pages QA'd divided be the personweeks of effort
PTL QA:	No. Pages	The number of pages QA'd by the PTL.	PTL	Monthly	If the Quality of the original work increases, the QA checks should go much faster, hence throughput of pages will be higher.	Pages QA'd divided be the personweeks of effort
Art Reviewed	No. Pieces	The number of pieces of Art reviewed by the PTL	PTL	Monthly	If the Quality of the original work increases, the art reviews should go much faster, hence throughput of pieces will be higher.	Pieces reviewed divided by the personweeks of effort
Documentation Rework Causes	No. Translate	No. pages reworked due to errors made by the translation vendor	LTL	Monthly	Knowing the amount and type of rework, we can concentrate on eliminating the major root causes of the rework	Pages reworked due to each category, as a percentage of the total reworked
	No. Format:	No. pages reworked by the Editor due to formatting errors.	PTL			
	No. Update:	No. pages reworked by the Localiser due to updates	LTL			
	No. Loc:	No. pages reworked by the Localiser due to Localisation error.	LTL			
Art Rework Causes	No. Format:	No. pieces reprepped by the Artist due to formatting errors in the prepping process	PTL	Monthly	Knowing the amount and type of rework, we can concentrate on eliminating the major root causes of the rework	Pieces reworked due to each category, as a percentage of the total reworked
	No. Update:	No. pieces reworked by the Artist due to updates	PTL			
	No. Loc:	No. pieces that were reshot due to errors in the original dump.	LTL			

Software

Measurement item	Metric	Definition	Who	When	Benefits	Useful calculations for reports
Effort - Person weeks	No. P/weeks Test	The no. of p/weeks Test Technicians spent working on testing each section of the Software - exe, setup, cbt, help, wizards and add-ins Also time spent by Engineers and Localisers in bugfixing effort	TTL	Monthly	Measuring effort spent helps improves planning, costing and scheduling of the project	
	No. P/weeks Engfix	Time spent by Engineers in bugfixing effort	SWM			
	No. P/weeks Locfix	Time spent by Localisers in bugfixing effort	LTL			
Total bugs		The TOTAL number of bugs in the database on the last week of the calendar month	TTL	Monthly	Number of additional bugs per month should decrease as the shipdate approaches. The total bugs per language should decrease as the releases proceed. Improving the localisation process should produce fewer total bugs over time.	Obtain from RAID
Bad bugs	No. Bad Bugs	The no. of bugs with resolved = not repro, duplicate or by design	TTL	Monthly		
No. Active bugs	No. Active bugs	The number of bugs with STATUS = Active on the last week of the calendar month	TTL	Monthly	Should decrease closer to ship date	Divide Active bugs by the Total number of bugs, expressed as a %
US bugs:	No. US bugs	The number of bugs found which also occur in the US version, and were not introduced during Localisation	TTL	End of Project		Obtain from RAID
International functionality:	No. Intl Funct. bugs	The number of bugs that were caused by the US code not being developed for ease of localisation	TTL	Monthly	This gives a measure of the localisability bugs remaining in the code	Obtain from RAID
Localisation	No. Localisation bugs	A localisation bug is one which was introduced during the localisation process.	TTL	Monthly	This shows us the number and % of bugs that were introduced during the localisation process. The number of localisation bugs should decrease.	Obtain from RAID

Measurement item	Metric	Definition	Who	When	Benefits	Useful calculations for reports
	No. bugs per thousand localisable items		TTL	Project end	This help us compare bug densities across projects and across PUs	Divide no. of localisation bugs by the number of localised items, (KItems) and express per thousand
Bug Causes	**No. hotkey:** **No. DBox:** **No. Text** **No. Int'l F**	Break down the total number of Localisation errors into these categories *Hotkey* (duplicate or missing) *Dialog box* (missizings; misalignments) *Text*: (missing; alignment, incorrect; untranslated, punctuation) *Int'l F*: (hard coding; macro func),	TTL	Project End to start with. Monthly once it is automated	This is one of the most important metrics and the bugs are further broken down into the most common categories. This helps us focus our improvement efforts on the main causes of these errors.	Bugs in each category, as a percentage of the total localisation bugs
Bug-fixing time:	Cost in time of each category of bug	Average elapsed time between activating and closing the bug report for each bug category	TTL	Project end	This shows us the amount of time being spent finding, fixing and regressing each category of bug, hence where most time savings can be made by reducing the number of certain types of error	Obtain from RAID
Test Pass:	Current test pass	This shows the current test pass number for each element, to include regression passes - .exe, setup, cbt, help, wizards and add-ins	TTL	Monthly	Ideally, there should be 2 test passes (the first one and the regression pass). From month to month, we can also see the increase in the total number of bugs per test pass.	For planning purposes, divide the no. of personweeks by the test pass no. to get average test pass duration for each element of the software
Unapplied Personweeks	No. P/weeks	No. weeks Testers were not working on the project	TTL	Monthly	Will help improve planning, costing and scheduling	

Measurement item	Metric	Definition	Who	When	Benefits	Useful calculations for reports
Effort - person weeks	No. p/weeks Localisation	The no. of localiser personweeks spent working on each section of the Software, as defined below	LTL & SWM	Monthly	Measuring effort spent has two main uses - firstly, it improves planning and scheduling of the project, secondly is used for the productivity metrics below	Personweeks used as basis for the other measures
Tokens translated:	No. tokens translated	The number of strings translated by the Localiser in the month.	LTL	Monthly	This gives us a ball-park figure of how long it takes to localise token files, and will measure the efficiency of new tools such as Glossman as compared to a fully manual process. The cumulative % complete also aids project management.	No. strings divided be the personweeks of effort; and cumulative strings as a % of total
Dialog boxes:	No. DBs and no. Hotkeys	The number of dialog boxes prepared in the month and the corresponding no. of hotkeys localised	LTL	Monthly	Again, improvements in the process should be reflected here. Software Localisers spend most of their time either translating tokens or sizing dialog boxes. The cumulative % complete also aids project management.	No. DBs divided be the personweeks of effort; and cumulative DBs as a % of total
CBT Lessons:	No. Screens	The number of CBT screens localised in the month	LTL	Monthly	New CBT procedures and tools may impact the efficiency with which CBTs can be localised. The cumulative % complete also aids project management.	No. screens divided by the personweeks of effort; & cum. screens as a % of total
Help Pages:	No. pages and no. words	The number of pages of Help prepared in the month and the corresponding number of words.	LTL	Monthly	New Help procedures and tools may impact the efficiency with which Helps can be localised, which needs to be measured. The cumulative % complete also aids project management.	No. pages divided be the personweeks of effort; and cumulative pages as a % of total
Add-ins:	No. P/weeks	The time taken, to localise each add-in (add-ins to be specified on the sheet)	LTL	Monthly	Add-ins cannot be measured in the same way as other elements, but take up a substantial amount of time, and should be thoroughly planned for.	Cumulative personweeks as a % of the planned personweeks
Unapplied P/weeks	No. P/weeks	No. weeks Localisers were not working on the project	LTL	as above	Will help improve planning, costing and scheduling	
Engineer P/weeks	No. P/weeks	No. weeks Engineers spent setting up the project and helping with the localisation	SWM	as above		
Project planned totals	No. tokens, pages, words,	The number of each item originally planned for the project	PM	project start		

Appendix G

Suggested categories for an effort-reporting system

Area	Localiser	Engineer	Test	Production	Tech Spec	PM	Total
Glossaries	0	n/a	n/a	n/a	n/a	n/a	
Main.exe	0	0	0	n/a	n/a	n/a	
Add-ins	0	n/a	0	n/a	n/a	n/a	
Help	0	n/a	0	n/a	n/a	n/a	
CBT/Wizards	0	n/a	0	n/a	n/a	n/a	
DTP	n/a	n/a	n/a	0	n/a	n/a	
Indexing/proofing	n/a	n/a	n/a	0	n/a	n/a	
Rework	0	n/a	n/a	0	n/a	n/a	
Vendor QA	0	n/a	0	0	0	n/a	
Vendor Project Mgmt.	n/a	n/a	n/a	n/a	0	0	
Vendor Support	0	0	0	0	0	0	
Vendor Training	0	0	0	0	0	0	
Internal Project Support	n/a	0	n/a	n/a	0	n/a	
Project Management	n/a	n/a	n/a	n/a	n/a	0	
Total							

This gives a total of 14 activity areas, with 35 possible combinations between all functions.

Note that there should be no 'other' category, 'vacation/sick time', etc. Also, time should be recorded in mandays, not in hours, then rolled up each month to give information in manweeks.

Appendix H - Current Metrics

Category	Description	Source	Reporting mechanism
Size:	Number localisable words in main .exe and associated .dlls	PM	Post Mortem
budget, forecast,	Number localisable menus and dialog boxes in main .exe, etc.	PM	Post Mortem
shipped & worked on	Number localisable words in Setup, Add-ins	PM	Post Mortem
	Number words Help	PM	Post Mortem
	Number pages Doc	PM	Post Mortem
	Number pieces Art	PM	Post Mortem
	Number words/screens CBT	PM	Post Mortem
	Number cartons	PM	Post Mortem
	Number covers/misc docs	PM	Post Mortem
Timeliness	Actual vs. planned absolute release date	PM	Post Mortem
	Actual Delta vs. planned Delta	PM	Post Mortem
Releases	Number products released	PM	Month report
	Number disks released	PM	Month report
	Number pages film released	PSM	Month report
	Number products re-released	PM	Month report
	Number disks re-released	PM	Month report
	Number pages film re-released	PSM	Month report
Cost vs forecast	total project cost	Calculated	Post-Mortem
& vs budget	overall project cost per word by element	Calculated	Post-Mortem
	internal costs	Calculated	Post-Mortem
	external (invoiced costs)	Calculated	Post-Mortem
	Invoiced cost of rework	Calculated	Post-Mortem
Quality	Number localisation bugs per bug origin	RAID	Post Mortem
	Number bugs, broken down by 'type' keyword	RAID	Post Mortem
	Number loc bugs per K loc words, broken down by severity	Calculated	Post Mortem
	Software quality rating	Subsidiary	Sub Eval Form
	CBT quality rating	Subsidiary	Sub Eval Form
	Help quality rating	Subsidiary	Sub Eval Form
	Documentation quality rating	Subsidiary	Sub Eval Form
	Packaging quality rating	Subsidiary	Sub Eval Form
Effort	Number days project management	RTS	Post Mortem
(report both monthly	Number days spent in glossary preparation	RTS	Post Mortem
and in Post Mortem)	Number days software localisation	RTS	Post Mortem
	Number days Documentation localisation	RTS	Post Mortem
	Number days help localisation	RTS	Post Mortem
	Number days CBT/Wizards localisation	RTS	Post Mortem
	Number days add-ins localisation	RTS	Post Mortem
	Number days Software Engineering	RTS	Post Mortem
	Number days documentation dtp	RTS	Post Mortem
	Number days documentation indexing/proofing	RTS	Post Mortem
	Number days Software Testing	RTS	Post Mortem
	Number days Help Testing	RTS	Post Mortem
	Number days CBT/Wizards Testing	RTS	Post Mortem
	Number days add-ins testing	RTS	Post Mortem
	Number days rework	RTS	Post Mortem

Appendix H - Current Metrics

	Number days Vendor Project Management	RTS	Post Mortem
	Number days Vendor Support	RTS	Post Mortem
	Number days Vendor Training	RTS	Post Mortem
	Number days Vendor QA	RTS	Post Mortem
	Number days internal project support	RTS	Post Mortem
Throughput	Number words Software localised per manmonth	Calculated	Post Mortem
	Number help pages localised per manmonth	Calculated	Post Mortem
	Number CBT screens localised per manmonth	Calculated	Post Mortem
	Number pages documentation released per editor manmonth	Calculated	Post Mortem
	Percentage re-use documentation pages	Calculated	Post Mortem
	Percentage re-use help pages	Calculated	Post Mortem
	Percentage re-use CBT pages	Calculated	Post Mortem
	Percentage re-use pieces Art	Calculated	Post Mortem
	Number of bugs found per Testing manmonth	Calculated	Post Mortem
	Number of test passes	TTL	Post Mortem
Rework	Number words/pages help reworked	PM	Post Mortem
	Number words/pages doc reworked	PM	Post Mortem
	Number words/screens CBT reworked	PM	Post Mortem
Vendor rating	Software	Vendor QA	Post-Mortem
	Software language rating	Vendor QA	Post-Mortem
	Software formatting rating	Vendor QA	Post-Mortem
	Software functionality rating	Vendor QA	Post-Mortem
	Documentation	Vendor QA	Post-Mortem
	Help language rating	Vendor QA	Post-Mortem
	Help formatting rating	Vendor QA	Post-Mortem
	Help functionality rating	Vendor QA	Post-Mortem
	Help	Vendor QA	Post-Mortem
	Documentation language rating	Vendor QA	Post-Mortem
	Documentation formatting rating	Vendor QA	Post-Mortem
	Documentation functionality rating	Vendor QA	Post-Mortem
	CBT / Wizards	Vendor QA	Post-Mortem
	CBT language rating	Vendor QA	Post-Mortem
	CBT formatting rating	Vendor QA	Post-Mortem
	CBT functionality rating	Vendor QA	Post-Mortem
	Language	Vendor QA	Post-Mortem
	Test	Vendor QA	Post-Mortem
	Engineering	Vendor QA	Post-Mortem
	DTP Skills	Vendor QA	Post-Mortem
	Project Management / communication	Vendor QA	Post-Mortem
	Timeliness	Vendor QA	Post-Mortem

Appendix I
Draft Post Mortem Template

Date:

Project Name:

Languages:

PM(s):

S/W Engineer(s):

Localisation Team:

Testing Team:

Production Team:

PPS:

Translation Vendors:

Contents

EXECUTIVE SUMMARY

Project Overview

Introduction paragraph to project.

Schedules

Explain variations in actual vs. original. What went well and what to change for next version, in bullet form

LANGUAGE	*Original RTM*	*Actual RTM*	*Original DELTA*	*Actual DELTA*
US			n/a	n/a

Cost

Explain variations in actual vs. budget and vs. forecast. Reference detailed costs, Appendix A.

	Budget	Forecast	Actual
Internal costs			
External costs			
Total Project			

Localisation strategy

Summary paragraph on strategy/methods used to localise the product, eg Vendors, in-house, some of each.

Project Mangement

Summary of project management issues

Vendors

Summary of vendor strategy, usefulness, quality, etc.

Resources

Comment on resources - machines and people.

Tools

what tools were used - how this affected quality, timeliness, cost - what to change for next version.

Summary of Recommendations for next project

- (Bullet points, in order of priority)
-
-
-
-

Project Management

Contact:

Summary

Intro paragraph here. Keep it brief and to-the-point. Include how work with Vendors went, etc.

COST

	Budget	Forecast	Actual
Cost per word S/W			
Cost per word Doc			
Cost per word CBT			
Cost per word Help			

Software

Printed Documentation

Online Documentation

Vendors

Vendor Project Management Ratings

Vendor Name	*Proj Mgmt*	*Timeliness*	*Commun.*	*Lang*	*Eng*	*Test*	*DTP*

Areas for improvement

{Issue #1}
Summary of issue

Recommendation or Action Item
Specific recommendation or action item for next project.

{Issue #2}
Summary of issue

Recommendation or Action Item
Specific recommendation or action item for next project.

Things that worked well

{Success #1}
Summary of successful practice

{Success #2}
Summary of successful practice

Documentation

Contact:

Summary

Intro paragraph, commenting on resources, schedules, tools.

Size measures

UE Element	Budget	Forecast	Localised	Reused	Reworked	Shipped
Number pages DOC						
Number pieces Art						
*Number words/pages Help						
*Number words/screens CBT						

* indicate whether words or pages are being quoted.

Printed documentation

Schedules

Vendors

Include Vendor overall documentation rating

Resources

Tools

Packaging

Areas for improvement

{Issue #1}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

{Issue #2}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

Things that worked well

{Success #1}

Summary of successful practice

{Success #2}

Summary of successful practice

Online documentation

Schedule

Vendors
Include vendor overall CBT and Help ratings

Resources

Tools

Areas for improvement

{Issue #1}
Summary of issue

Recommendation or Action Item
Specific recommendation or action item for next project.

{Issue #2}
Summary of issue

Recommendation or Action Item
Specific recommendation or action item for next project.

Things that worked well

{Success #1}
Summary of successful practice

{Success #2}
Summary of successful practice

Software

Contact:

Summary

Summary paragraph should contain details of the compile kit, tools, and interaction with other groups in WPGI and Redmond. Keep to 1 page max.

Size measures

Software Element	Budget	Forecast	Actual
Words in Main.exe			
Menus in Main .exe			
Dialogs in Main .exe			
Words in Setup/Add-ins			
Words in Text files			

Localisation/Vendors

Schedule

Vendors

Include vendor overall software rating

Resources

Tools

Engineering

Schedule

Resources

Tools

Areas for improvement

{Issue #1}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

{Issue #2}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

Things that worked well

{Success #1}

Summary of successful practice

{Success #2}

Summary of successful practice

Testing

Contact:

Summary

Start with a summary paragraph of the testing strategy, and comment on interaction with localisation and the US Test group, automation, resources and schedules.

Schedule

Resources

Tools

Include summary of automation strategy.

Quality

Reference pareto charts and bug density metrics in Appendix B.

Software

Language						
Resource fix						
Code-fix						
Compile Error						
US Bug						
Total						

Include summary of software quality and conclusions from bugs info above

Add-ins

Language						
Resource fix						
Code-fix						
Compile Error						
US Bug						
Total						

Include summary of add-ins quality and conclusions from bugs info above

Help

Language						
Resource fix						
Code-fix						
Compile Error						
US Bug						
Total						

Include summary of help quality and conclusions from bugs info above

CBT/Wizards

Language						
Resource fix						
Code-fix						
Compile Error						
US Bug						
Total						

Include summary of CBT/Wizards quality and conclusions from bugs info above

Bugs per Tester day (Testing efficiency)

divide number of bugs by number of tester days

Language						
Software						
Add-ins						
Help						
CBT						
Total						

Areas for improvement

{Issue #1}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

{Issue #2}

Summary of issue

Recommendation or Action Item

Specific recommendation or action item for next project.

Things that worked well

{Success #1}

Summary of successful practice

{Success #2}

Summary of successful practice

Appendix A - Project Costs

(attach Excel Spreadsheet containing the following info, per language

Category	Software	Documentation	Help	CBT	Total
Total cost					
Invoiced rework					

General observations and conclusions of Cost measures:

-
-
-

Appendix B - Quality Measures

Pareto Chart of Bug Type (per language)

(import this from an Excel Chart/Spreadsheet, having obtained the info through a RAID query)

Bug Density

Number of bugs per thousand localisable words

(divide number of localisation bugs by number of localised words for each of software, add-ins, CBT and Help, and multiply by one thousand. Note, setup may be included with add-ins, or different types of add-in may be separated. It may also be useful to separate severity 1/2 bugs from severity 3/4 bugs)

Language						
Software						
Add-ins						
Help						
CBT						
Total						

General observations and conclusions of Quality measures:

- (bullet form)
-
-
-

Appendix C - Effort measures

Attach spreadsheet or charts containing the effort measures for eacg project category

General observations and conclusions of Effort measures:

- (bullet form)
-
-
-

Appendix D - Internal efficiency metrics

Attach spreadsheet containing the following info, which has been calculated from the software size and the total effort.

Internal Efficiency Stats	Software Words	Help pages	CBT screens	Doc pgs. reviewed
No. per person-month				

Appendix E - Vendor Localisation Ratings

Attach the following information in the form of a report from the vendor database.

Vendor Name	*Language*	*Formatting*	*Funct.*	*Vendor Name*	*Language*	*Formatting*	*Funct.*
Software				Software			
Documentation				Documentation			
Help				Help			
CBT				CBT			
Packaging				Packaging			

Vendor Name	*Language*	*Formatting*	*Funct.*	*Vendor Name*	*Language*	*Formatting*	*Funct.*
Software				Software			
Documentation				Documentation			
Help				Help			
CBT				CBT			
Packaging				Packaging			

Vendor Name	*Language*	*Formatting*	*Funct.*	*Vendor Name*	*Language*	*Formatting*	*Funct.*
Software				Software			
Documentation				Documentation			
Help				Help			
CBT				CBT			
Packaging				Packaging			

General observations and conclusions of Vendor measures:

- (bullet form)
-
-
-

BIBLIOGRAPHY

[Albrecht83] *Software function, source lines of code, and development effort prediction: a software science validation*, Allan J. Albrecht and John E. Gaffney Jr., IEEE Transactions on Software Engineering, Vol. SE9, No. 6, pp. 639-648, November 1983.

[AMI92] *Metric Users' Handbook*, The AMI Consortium, South Bank Polytechnic, London, England, 1992

[Ashley91] *Measurement as a Management Tool*, Nicholas Ashley, proceedings of seminar 'Management and Testing of Software for Quality', Unicom Seminars Ltd., London, Feb. 1991

[Beizer84] *Software System Testing and Quality Assurance*, Boris Beizer, Van Nostrand Reinhold, New York, 1984

[Bhide90] *Generalised Software Process-Integrated Metrics Framework*, Sandhiprakash Bhide, Journal of Systems and Software, vol. 12: pp 249-254, 1990

[Bock92] *FP-S: A Simplified Function Point Counting Method*, Douglas B. Bock and Robert Klepper, Journal of Systems and Software, Vol. 18, pp245-254, 1992

[Boehm81] *Software Engineering Economics*, B.W. Boehm, Prentice-Hall, Englewood Cliffs, New Jersey, 1981

[Bollinger91] *A Critical look at Software Capability Evaluations*, Terry Bollinger and Clement McGowan, IEEE Software, July, 1991

[Brooks75] *The Mythical Man-Month*, Frederick P. Brooks, Addison Wesley, 1982

[Choppin91] *Quality through People*, Jon Choppin, IFS Publications, London, 1991

[Cusumano91] *Japan's Software Factories*, Michael A. Cusumano, Oxford University Press, New York

[CSE92] Proceedings of the *Colloquium on Cost Estimation*, 31 March, 1992, Centre for Software Engineering, Dublin City University.

[Davis92a] *Develop applications on time, every time*, Dwight B. Davis, Datamation,, v38, n22, pp85-88 Nov 1, 1992

[Davis92b] *Does your IS shop measure up?*, Dwight B. Davis, Datamation, v38, n18, pp 26-32, Sept. 1, 1992

[Dehnad90] *Software Metrics from a User's perspective*, Khrosrow Dehnad, J. Systems Software 1990; 13: pp 111-115

[DeMarco82] *Controlling Software Projects*, T. De Marco, Yourdon Press, Englewood Cliffs, New Jersey, 1982

[Green92] *Fewer programmers produce fewer bugs*, Robert Green, Government Computer News, v11 n16 p81, Aug 1992

[Halstead77] *Elements of Software Science*, M. H. Halstead, Elsevier N-Holland, 1977

[Henry90] *Integrating Metrics into a Large-Scale Software Development Environment*, Sallie Henry and John Lewis, J. Systems Software 1990; 13: pp 89-95.

[Hooten92] *Management By Numbers*, Karen Hooten, Computer Language, Dec 1992, v9 n12 p93, Miller Freeman Publications Inc.

[Humphrey88] *Characterising the Software Process: A Maturity Framework.* Watts S. Humphrey, IEEE Software, March 1988.

[Humphrey91a] *Comments on 'A Critical Look'*, Watts S. Humphrey and Bill Curtis, IEEE Software, July 1991

[Humphrey91b] *Software process Improvement at Hughes Aircraft,* Watts S. Humphrey et al, IEEE Software, July 1991

[Jones85] *A process-integrated approach to defect prevention,* Carole L. Jones, IBM Systems Journal, Vol. 24, No. 2, 1985

[Kadota92] *Software Development Management Based On Progress Curves Categorisation*, Yasuhiro Kadota et al, proceedings of the 3rd European Conference on Software Quality, Madrid, November 1992

[Kemerer93] *Reliability of Function Points Measurement - A Field Experiment,* Chris F. Kemerer, Communications of the ACM, Vol. 36, No. 2, February 1993

[Mermaid91] *Description of the Knowledge Base*, deliverable ref: D4A, MERMAID - ESPRIT project P2046, 1991

[Metkit92a] *Case Study - Setting up a Measurement Programme* - Metkit/ICSP/SN/ISSUE2, Esprit project 2384, Metkit Consortium, 1992.

[Metkit92b] *Defect Analysis as an Improvement Tool* - Metkit/IDFA/SN/ISSUE2, Esprit project 2384, Metkit Consortium, 1992.

[McCabe76] *A Complexity Measure*, T. McCabe, IEEE Transactions on Software Engineering, Vol. SE2, No. 4 pp. 308-320, December 1976.

[Musa89] *Quantifying Software Validation: When to stop Testing?*, John D. Musa, IEEE Software, May 1989

[Oates92] *Quality Starts at specs: open files,* Joe Oates, Software Magazine, vol. 12, no. 11, pp6-7, August, 1992

[Peeters89] *Thriving on Chaos*, Tom Peeters, Pan Books, London, 1989

[Pfleeger90] *Software Metrics in the Process Maturity Framework*, Shari Lawrence Pfleeger and Clement McGowan, Journal of Systems and Software, Vol. 12, pp255-261, 1990

[Pyramid91] *Quantitative Management: get a grip on software!*, PYRAMID Consortium, ESPRIT project EP5425, 1991

[Qualtec91] *Planning and Implementing TQM* (Participant Workbook), Qualtec Inc., 1991

[SEI91a] *Software Project Effort and Schedule Measurement* (Draft), Carnegie Mellon University, 1991

[SEI91-TR16] *Measurement in Practice*, Stan Rifkin and Charles Cox, CMU/SEI-91-TR-16, Carnegie Mellon University, 1991

[SEI91-TR24] *Capability Maturity Model for Software*, M. C. Paulk et al, CMU/SEI-91-TR-24, Carnegie Mellon University, 1991

[SEI91-TR25] *Key Practices for the Capability Maturity Model*, C. V. Weber et al, CMU/SEI-91-TR-25, Carnegie Mellon University, 1991

[SEI92-TR19] Software Measurement for DoD Systems: Recommendations for initial core measures, Anita D. Carleton et al, CMU/SEI-92-TR-19, Carnegie Mellon University, 1992

[SEI92-TR20] *Software Size Measurement: A Framework For Counting Source Statements*, R. E. Park et al, CMU/SEI-92-TR-20, Carnegie Mellon University, 1992

[SEI92-TR21] *Software Effort Measurement: A Framework for Counting Staff Hours*, W. B. Goethert et al, CMU/SEI-92-TR-21, Carnegie Mellon University, 1992

[SEI92-TR22] *Software Quality Measurement: A Framework for Counting Problems and Defects*, William A. Florac, CMU/SEI-92-TR-22, Carnegie Mellon University, 1992

[SEI92-TR24] *An Analysis of SEI Software Process Assessment Results: 1987-1991;* D.H. Kitson, S. Masters, CMU/SEI-92-TR-24, Carnegie Mellon University, 1992

[SEI92-TR25] *Software Measures and the Capability Maturity Model*; J. H. Baumert, M. S. McWhinney, CMU/SEI - 92-TR-25, Carnegie Mellon University, 1992

[Sellers92] *Technical Correspondence*, B. Henderson Sellers, Communications of the ACM, Vol. 35, No. 12, December, 1992

[Shen83] *Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support*, Vincent Y. Shen et al, IEEE Transactions on Software Engineering, Vol. SE9, No. 2, pp 155-165, March, 1983

[Sprouls90] *IFPUG Function Point Counting Practices Manual Release 3.0*, J. Sprouls, International Function Point Users Group, Westerville, Ohio, 1990

[Walsh93] *Determining Software Quality*, James Walsh, Computer Language, Vol. 10, no. 4, pp 57-62, April 1993

[Ward89] *Software Defect Prevention Using McCabe's Complexity Metric*, William T. Ward, Hewlett-Packard Journal, April 1989.