**ANECON**

**System Measurement Process**

The SoftMess Measurement Workbench

For Auditing, Controlling, Evaluating & Calculating Software Products & Projects

Version: 2.1

Datum: March, 2011

|  |
| --- |
| Harry M. Sneed |
| [Harry.Sneed@anecon.com](mailto:Harry.Sneed@anecon.com) |



Software Design und Beratung G.m.b.H.

Alserstraße 4 / Hof 1

A-1090 Wien

[The SoftMess Measurement Workbench 5](#_Toc292979826)

[1. Auditing and Measuring Software Products with SoftAudit 6](#_Toc292979827)

[1.1 Auditing Software Products 6](#_Toc292979828)

[1.1.1 Auditing Requirement Documents 7](#_Toc292979829)

[1.1.2 Auditing Design Models 8](#_Toc292979830)

[1.1.3 Auditing Program Code 8](#_Toc292979831)

[1.1.4 Auditing Database Schemas 9](#_Toc292979832)

[1.1.5 Auditing Data Interfaces 9](#_Toc292979833)

[1.1.6 Auditing Test Cases 10](#_Toc292979834)

[1.2 Measuring Software Products 10](#_Toc292979835)

[1.2.1 Measuring Software Quantity 11](#_Toc292979836)

[1.2.1.1 Measuring Requirement Document Size 11](#_Toc292979837)

[1.2.1.2 Measuring Design Model Size 12](#_Toc292979838)

[1.2.1.3 Measuring program size 12](#_Toc292979839)

[1.2.1.4 Measuring database size 12](#_Toc292979840)

[1.2.1.5 Measuring interface size 13](#_Toc292979841)

[1.2.1.6 Measuring test case quantity 13](#_Toc292979842)

[1.2.2 Measuring Software Complexity 13](#_Toc292979843)

[1.2.2.1 Measuring Requirements Complexity 13](#_Toc292979844)

[1.2.2.2 Measuring Design Complexity 13](#_Toc292979845)

[1.2.2.3 Measuring Program Complexity 14](#_Toc292979846)

[1.2.2.4 Measuring Database Complexity 14](#_Toc292979847)

[1.2.2.5 Measuring Interface Complexity 15](#_Toc292979848)

[1.2.2.6 Measuring Test Case Complexity 15](#_Toc292979849)

[1.2.3 Measuring Software Quality 15](#_Toc292979850)

[1.2.3.1 Requirement Quality Measurement 16](#_Toc292979851)

[1.2.3.2 Design Quality Measurement 16](#_Toc292979852)

[1.2.3.3 Program Quality Measurement 17](#_Toc292979853)

[1.2.3.4 Database Quality Measurement 17](#_Toc292979854)

[1.2.3.5 Interface Quality Measurement 18](#_Toc292979855)

[1.2.3.6 Test Case Quality Measurement 18](#_Toc292979856)

[1.2.4 Aggregating the Measurements 18](#_Toc292979857)

[2 Collecting Productivity Data with SoftProd 20](#_Toc292979858)

[2.1 Obtaining Data from Electronic Forms 20](#_Toc292979859)

[2.1.1 Processing Error Reports 21](#_Toc292979860)

[2.1.2 Processing Change Requests 22](#_Toc292979861)

[2.1.3 Processing Time Sheets 22](#_Toc292979862)

[2.2 Compiling Productivity Data 23](#_Toc292979863)

[2.3 Producing Productivity Reports 23](#_Toc292979864)

[2.3.1 Defect Metric File 24](#_Toc292979865)

[2.3.2 Change Metric File 24](#_Toc292979866)

[2.3.3 Time Metric File 24](#_Toc292979867)

[3 Evaluating Software Products with SoftEval 25](#_Toc292979868)

[3.1 Collecting and Storing Metric Data 26](#_Toc292979869)

[3.1.1 Storing Requirements Metrics 26](#_Toc292979870)

[3.1.2 Storing Design Metrics 26](#_Toc292979871)

[3.1.3 Storing Source Code Metrics 27](#_Toc292979872)

[3.1.4 Storing Data Metrics 27](#_Toc292979873)

[3.1.5 Storing Interface Metrics 27](#_Toc292979874)

[3.1.6 Storing Test Case Metrics 27](#_Toc292979875)

[3.1.7 Storing Test Data Metrics 27](#_Toc292979876)

[3.1.8 Storing Test Object Metrics 27](#_Toc292979877)

[3.1.9 Storing Defect Metrics 28](#_Toc292979878)

[3.1.10 Storing Change Metrics 28](#_Toc292979879)

[3.1.11 Storing Time Metrics 28](#_Toc292979880)

[3.2 Editing Metric Data 28](#_Toc292979881)

[3.2.1 Requirement Metric Table 28](#_Toc292979882)

[3.2.2 Design Metric Table 29](#_Toc292979883)

[3.2.3 Code Metric Table 29](#_Toc292979884)

[3.2.4 Data Metric Table 29](#_Toc292979885)

[3.2.5 Interface Metric Table 30](#_Toc292979886)

[3.2.6 Test Case Metric Table 30](#_Toc292979887)

[3.2.7 Test Data Metric Table 30](#_Toc292979888)

[3.2.8 Test Object Metric Table 30](#_Toc292979889)

[3.2.9 Defect Metric Table 31](#_Toc292979890)

[3.2.10 Change Metric Table 31](#_Toc292979891)

[3.2.11 Time Metric Table 31](#_Toc292979892)

[3.3 Producing Metric Reports 31](#_Toc292979893)

[3.3.1 Metric Tables 32](#_Toc292979894)

[3.3.2 Ranking Reports 33](#_Toc292979895)

[3.3.3 Comparison Reports 33](#_Toc292979896)

[3.4 Providing Graphic Displays of Metric Data 33](#_Toc292979897)

[3.4.1 Bar Charts 33](#_Toc292979898)

[3.4.2 Pie Charts 34](#_Toc292979899)

[3.4.3 Fishbone Diagrams 34](#_Toc292979900)

[3.4.4 Distribution Curves 34](#_Toc292979901)

[3.5.5 Manager’s Dashboard 34](#_Toc292979902)

[4 Calculating Software Projects with SoftCalc 36](#_Toc292979903)

[4.1 Functions of the SoftCalc tool 37](#_Toc292979904)

[4.1.1 Defining a Software Project 37](#_Toc292979905)

[4.1.2 Importing a Productivity Table 38](#_Toc292979906)

[4.1.3 Importing System Metrics 38](#_Toc292979907)

[4.1.4 Adapting system metrics 39](#_Toc292979908)

[4.1.5 Prescribing System Quality 39](#_Toc292979909)

[4.1.6 Setting Project Influences 39](#_Toc292979910)

[4.1.7 Rating Project Resource Factors 40](#_Toc292979911)

[4.1.8 Analyzing Project Risks 40](#_Toc292979912)

[4.1.9 Calculating Project Costs and Time 40](#_Toc292979913)

[4.1.10 Projecting Error Rates 40](#_Toc292979914)

[4.1.11 Predicting Maintenance Costs 41](#_Toc292979915)

[4.2 Inputs to SoftCalc 41](#_Toc292979916)

[4.2.1 SoftCalc User Interface 41](#_Toc292979917)

[4.2.1.1 Project / Product Table 41](#_Toc292979918)

[4.2.1.2 Object Table 42](#_Toc292979919)

[4.2.1.3 Use-Case Table 42](#_Toc292979920)

[4.2.1.4 Interface Table 42](#_Toc292979921)

[4.2.1.5 Test Case Table 43](#_Toc292979922)

[4.2.1.6 Component Table 43](#_Toc292979923)

[4.2.1.7 System Quality Table 43](#_Toc292979924)

[4.2.1.8 Project Influence Table 43](#_Toc292979925)

[4.2.1.9 Project Resource Table 43](#_Toc292979926)

[4.2.1.10 Project Risk Table 44](#_Toc292979927)

[4.2.1.11 Project Productivity Table 44](#_Toc292979928)

[4.2.2 SoftCalc Import Files 44](#_Toc292979929)

[4.2.2.1 Productivity File (ProdTab) 44](#_Toc292979930)

[4.2.2.2 Requirement Entity File (REntity) 44](#_Toc292979931)

[4.2.2.3 Design Entity File (UEntity) 45](#_Toc292979932)

[4.2.2.4 Code Entity File (CEntity) 45](#_Toc292979933)

[4.2.2.5 Data Entity File (DEntity) 45](#_Toc292979934)

[4.2.2.6 Interface Entity File (IEntity) 45](#_Toc292979935)

[4.2.2.7 Test Case File (TEntity) 46](#_Toc292979936)

[4.2.2.8 Test Data File (XEntity) 46](#_Toc292979937)

[4.2.2.9 Test Protocol File (PEntity) 46](#_Toc292979938)

[4.3 Outputs from SoftCalc 46](#_Toc292979939)

[4.3.1 Displays 46](#_Toc292979940)

[4.3.2 Printed Reports 47](#_Toc292979941)

[4.3.3 Exports 48](#_Toc292979942)

[4.4 Using SoftCalc 48](#_Toc292979943)

# The SoftMess Measurement Workbench

Harry M. Sneed

Vienna in March 2011

Measuring and evaluating existing software products is essential to planning and controlling their evolution. Without adequate information it is not possible to make rational decisions about what to make with the product, for instance whether to reengineer it, to migrate it, to integrate it or simply to go on maintaining it. Before starting a reengineering or refactoring project the responsible manager should know what the current quality of the product is, otherwise he cannot measure the degree of improvement. Before launching a migration project the migration manager must know the size, the complexity and the quality of the product, otherwise he cannot plan the migration. The same applies to wrapping and integrating existing software. The responsible managers need exact figures to determine whether the integration is feasible and if so, what it will cost. Even if a user decides to keep his current system and to go on maintaining it as it is, the maintenance tasks still have to be planned and monitored, and for that data is needed, data on the state of the software product as well as data on the previous maintenance projects. The planning of future evolution projects presupposes that data on past such projects is available. Acquiring such data in a timely and accurate manner is a task which only can be performed in today’s complex IT-world with the help of automated measurement tools.

The measurement tools offered by Harry M. Sneed in cooperation with the SORING Ltd. of Budapest, Hungary and the ANECON GmbH, Vienna perform four essential tasks in connection with the assessment and evaluation of software products as well as for the control and calculation of software projects:

* auditing and measuring software products including requirements, designs, source code, databases and interfaces with the tool *SoftAudit*,
* compiling productivity data, i.e. process metrics, with the tool *SoftProd,*
* aggregating metrics and evaluating software products and processes with *SoftEval*,
* estimating the time and costs of software projects with *SoftCalc*. *(See Figure 1: The SoftMess Software Measurement Workbench).*



# Auditing and Measuring Software Products with SoftAudit

The basis for evaluating software products and for estimating software projects are the measurements taken of the software. In principle all software artifacts can be measured including the requirements, the design models – both data and function models, the database structures, the interface definitions, the test cases and the source code. The tool for measuring all of these software artifacts is the tool ”*SoftAudit*”, which not only measures the software artifacts, but also checks those artifacts against the rules for constructing them. Software here is understood as the source code of a system as well as all of the documents describing the system. It is expressed in a selected language such as English for the requirements, UML for the design, Java for the program code, XML for the interface code and SQL for the database structure. The software world is abundant with languages. There are 2nd, 3rd, 4th and even 5th Generation languages, procedural and object-oriented languages, typed and untyped languages. For any given language there should be a set of rules and conventions for using that language. Part of a quality assessment is to determine to what extend software complies with the prevailing rules and conventions, laid down by the developing organization. Auditing, i.e. rule checking is very closely associated with measurement, counting language attributes and making calculations with them, which is why the two activities are combined in one tool.

As a result of the dependencies referred to here, the Software Auditor performs two functions

* The one is to check the software for compliance to standards,
* The other is to measure the software product.

## Auditing Software Products

Auditing software implies checking the software for compliance with existing conventions. There are different standards and conventions for the different types of software artifacts. There should be at least six:

* Convention for requirement documents
* Convention for design models
* Convention for program code
* Convention for database schemas
* Convention for system interfaces
* Convention for system test cases.

The basic standard for requirement specification is the IEEE–830 Guide to Requirement Specification. In addition there is a standard from the OMG and several domain specific conventions.

The basic standards for software design are the IEEE–1016, IEEE–1320 and the UML - a standard of the OMG. The UML standard has been adapted by the IEEE and can now be considered a universal standard for design documentation.

For programming languages there are many guidelines. In fact almost every book about a programming language also includes a guideline to using that language. Scott Ambler has written guidelines for using C++ and Java. These guidelines contain explicit rules on which statements to avoid and how other statements should be formulated. Similar coding conventions have been written for COBOL and PL/1 by this author. Steve MacConnell from Microsoft has written a guideline for using C#. There are also guidelines to using PHP, XSL and Java’s Script. So there is no lack of conventions for writing code. What is lacking is the enforcement of these conventions. Coding standards not only reduce the error probability, they also increase readability, interoperability, reusability and maintainability. Above all, they ensure that all software components are structured and coded in a similar way. A software development shop without coding standards is like a city without traffic rules. Everyone defines for himself what rules he wants to follow, if any. The systems can somehow function, just as Naples functions without traffic rules, but persons who are used to living in an ordered world will feel uncomfortable there.

Most database schemas are now coded in some form of SQL. The problem is that every database supplier has his own version of SQL. Every database supplier provides guidelines for setting up his particular database system. These include rules for coding the database schemas. These rules recommend some options and set some limits, but they need to be enhanced by others rules to ensure the quality of the data, rules such as limits to the size of the tables and the number of relationships. There is now a growing concern about the quality of data and a number of books and papers on how to ensure that quality.

System interfaces include external end user interfaces such as GUIs, web pages and masks, but also internal program to program and system to system interfaces. The former are coded on the main frame in some screen description language like CICS-BMS or IMS-MFS. In client server systems they were coded in 4GL languages like PowerBuilder and Delphi or with Java swing classes. In modern web applications the web pages are coded in HTML, XHTML, XSL or some script language like Flex or PHP. Thus, the rules for implementing user interfaces depend very much on the language being used. There are, however, some general rules like restricting the number of widgets on the screen and excluding processing logic from the screen description, which apply to all user interface descriptions.

The internal system interfaces are for passing data between components and systems. They were originally simple call interfaces like RPCs and RMIs, but later these were formalized in an interface definition language – IDL. Since the emergence of XML as a universal data exchange language, XML and XML derivates are used, such as WSDL for communicating with web services. The literature offers little on how to use these interfaces, but the author of these tools has been working on this subject for several years and has established a set of rules for coding interfaces based on his experience.

The only universally acknowledged standard for software test is the IEEE–829 which describes which test documents are to be produced and what content they should have. Unfortunately, there is no universal language for specifying test cases. Every test tool vendor has his own language. However, as a rule, most test cases are documented in excel or CSV type tables where every test case attribute is assigned to a column of that table. Some attributes are required, others are optional. So it is at least possible to check if the required attributes are present and that all attributes of a test case are consistent with one another.

## Auditing Requirement Documents

The auditing of the requirement documents is done by the tool „SoftAudit” which contains a text analyzer for both English and German. Besides controlling adherence to the IEEE–830 standard on the contents of a requirements document, it also checks compliance with the Rupp rules for formulating requirement sentences. There are rules governing what a requirements document should contain and rules for writing them. Of course many of these rules are rather vague. Since requirement documents are by nature informal, it is clear that the rules for writing them are also informal. Nevertheless the German and English text auditing tools have some 25 structural rules and 10 style rules for ensuring the quality of text documents.

## Auditing Design Models

Design documents are more formal than requirement documents but not as formal as code. Therefore there are more possibilities for checking them. The *SoftAudit* family of tools contains a tool for auditing UML models with the name UMLAudit. It processes the XMI files behind the UML diagrams. The XML syntax used should be common for all UML tools but it is not. There are slight variations from tool to tool. The rules are not intended to ensure that the UML model is correct. They can only ensure that the model is complete and consistent, that the proper names are used and that the design model remains within the limits of what can be readily understood and managed. For this there are currently 20 rules.

## Auditing Program Code

The *SoftAudit* tool set was originally intended for the auditing of source code. The problem here is that there are so many different programming languages. There must be a separate auditor for each language. Currently there is an auditor for:

The legacy mainframe languages:

* Assembler
* PL/1
* COBOL

The fourth generation languages:

* APS
* CSP
* Delta
* Natural
* ABAP
* PowerBuilder

The object-oriented languages:

* C/C++
* C#
* Java.

To audit the programs there is a recommended list of rules for each programming language. These rules are listed out in a rule table. The user has the possibility of deleting rules which he does not want to apply, however he cannot add rules. The rule checking is made on a statement by statement basis. Statements which violate one or more rules are listed out in the deficiency report. The user also has the possibility of setting certain size limits such as the number of statements per module, the number of methods per class and the maximum procedural nesting level. The program auditor will check whether these limits are exceeded and if so it will report the rule violation. The limits can, of course be different, from language to language.

The rule violations are classified as being major, minor or medium. Major rule violations are those which can lead to an error or a security leak. Medium rule violations are those which detract from the maintainability, portability, reusability, convertibility, etc. of the code. Minor rule violations are style deficiencies, which lower readability. The relationship of weighted code deficiencies to the number of statements determines the conformity measurement of the code.

## Auditing Database Schemas

There still exist legacy databases alongside the relational databases. Legacy databases can be either hierarchical or networked. IMS from IBM is the most common hierarchical database still in use. IDMS and IDS are representative of the networked databases. ADABAS from the Software AG is somewhere in between the networked and the new relational databases. Each of these legacy databases has its own special database description language. Thus, each requires its own rules.

The relational databases use a common language to describe the structure of the data – SQL. But here too each database vendor has his own unique variation of SQL. The rules may be the same but they have to be adjusted. Not all rules apply to all relational database types.

There is a database schema auditor for the following database types

* DLI (IMS)
* DDL (IDMS/IDS)
* ADABAS
* SQL (DB2/Oracle/Ms)

Just as with programs, databases should be defined in such a way as to conform to prevailing standards. Here too, the rules are stored in a table and displayed to the user for selection. The type and number of rules depends on the type of database description being audited, whether it is hierarchical, networked or relational. The user may delete rules, but he may not add new ones. There are also limits to be checked such as the maximum number of attributes per table, the maximum number of indexes and the maximum field length.

The database rule violations are classified as major, medium and minor. Major rule violations can cause performance problems or make it difficult to access the database. Medium rule violations are those where size limits are exceeded. Minor rule violations are invalid data types and missing attributes. All rules violations are recorded in the database deficiency report. The database conformity is the relationship of the weighted rule violations to the number of lines in the database schema.

## Auditing Data Interfaces

There are even more different data interface languages. They vary from ancient IMS-MFS maps to modern WSDL web service interfaces. It would be impossible to cover them all. Currently, there are interface definition auditors for each of the following interface types:

* BMS Maps
* MFS Maps
* Natural Maps
* IDL Interfaces
* XML,XSL,XSD Definitions
* WSDL Interfaces
* HTML Pages

The auditing of interfaces is a new feature of software quality assurance. There are interfaces to the user like HTML forms, BMS maps and XSL style sheets. There are also interfaces between systems such as XML, IDL and WSDL. The fact is that interfaces should also conform to local standards since interfaces in particular represent how a system is perceived from outside. For compatibility reasons only certain data types should be allowed. There should also be no redefinitions and no invisible data fields. There are different rules for different interface description languages. It is up to the user to adjust the rules to his requirements.

As with programs and databases there are also limits to a data interface. Screens should not be cluttered with widgets, messages should not contain too many parameters and there should be limits to the structural depth of XML trees. The more information an interface contains, the harder it is to test. There is a direct relationship between the width of an interface and its testability. These and other limits can be set by the user and they will be checked. All rule violations are recorded in the interface deficiency report and the degree of conformity computed from the relationship between the weighted number of rule violations and the number of interface description statements.

## Auditing Test Cases

Test cases can be described in many ways, as prosa text, as structured forms, as tables and in a formal language. TTCN-3 is a formal language for specifying test cases just as is TCS and the DataTest Assertion Language. These languages can be parsed, analyzed and checked against a set of rules. However, in the IT Praxis most test cases are stored in tables, either Excel type tables or in relational database tables. Each user can assign different attributes to a use case. Therefore in order to check them, the user has to first define what each column of his test case table means. He must also define which attributes are mandatory and which not. Once this is done, the test cases can be audited.

The SoftAudit tool family contains a tool for auditing testcases – TCSAudit. TCSAudit checks the test case tables, that all of the mandatory attributes are defined and that their values are valid. It also checks the test cases against the guidelines specified by the ANSI – 829 Standard and ensures that they are consistent with the requirements extracted from the requirements document. Every test case must be associated with a particular requirement, object or use case. Inversely every requirement, object and use case must have at least one test case. This is part of the consistency check made by TCSAudit.

## Measuring Software Products

There are three dimensions of software to be measured

* quantity or size,
* complexity and
* quality.

*(See Figure 2: Auditing and Measuring Text Artifacts)*



## Measuring Software Quantity

Software quantity can be measured in terms of different size measurements. Typical size measurements are among others

* the number of code lines
* the number of code statements
* the number of data variables
* the number of methods or procedures
* the number of classes or modules
* the number of interfaces
* the number of components
* the number of files
* the number of database tables etc.

What size measures are used depends on the type of software artifact being measured. Requirement documents contain sections, sentences, nouns, verbs, states, actions, conditions, etc. Design documents contain model types, entities and relationships. Procedural programs have different characteristics than object–oriented ones. User interfaces in the form of maps, panels, GUIs, and web pages have their own size measures such as templates, widgets and fields, just as do system interfaces like IDL, XML, and WSDL with their parameters, operations and links. Databases have again their own size dimensions in terms of their attributes, keys and indexes. There is no one universal size measurement other than perhaps lines of code or text.

To make software sizing more universal the elementary size counts such as those mentioned are compounded into complex size measures. This is the case with data–points, function–points and object–points. These sizes are computed by aggregating the elementary size measures such as the number of inputs, outputs and files or classes, attributes and methods. The result is a complex size metric which can be used for comparing the size of various software systems in different languages.

## Measuring Requirement Document Size

Requirement documents should be structured texts with defined textual entities such as business goals, business processes, business objects, business rules, business interfaces, business requirements and use cases. These business entities can be further broken down into logical attributes such as identifiers, data attributes, triggers, states, actions, preconditions and post conditions. In measuring the size of the projected software system based on the requirements, it is necessary to identify and count the business entities, their attributes and their relationships. These elementary measures can then be combined to compute complex size metrics like function-points and use case-points. For measuring the size of the requirements SoftAudit has no less than 32 elementary counts and 5 complex size metrics use-case points, function-points, data-points, object-points and test-points.

## Measuring Design Model Size

Design documents are usually parts of an overall design model. Thus there are documents describing data flow, documents describing control flow, documents describing the data structure and other documents describing interactions among procedures. The UML language contains a number of well defined model types which are combined to produce design documents. A design document is actually a view of the system architecture. To measure the size of a design as a whole, all of the model types have to be recognized and counted. For instance, one must count the number of objects, the number of object states and the number of state transitions. Here too general size metrics like data-points and object-points can be deduced from the elementary model type counts. This should lead to an even more accurate size measure of the software system than that taken from the requirements, but only if the design model is really complete. To measure the size of a design model, SoftAudit has 28 entity counts and 10 relationship counts plus the 5 complex size metrics - Data-Points, Object-Points, Function-Points, UseCase-Points and Test-Points.

.

## Measuring program size

For measuring the size of programs *SoftAudit* takes no less than 56 different elementary size measures:

* 7 code quantity measures
* 9 structural quantity measures
* 18 data quantity measures
* 22 procedural quantity measures

From these elementary counts 3 compound size metrics are derived

* data–points,
* functions–points and
* object–points.

## Measuring database size

For measuring databases *SoftAudit* takes some 32 elementary size measures such as number of keys, number of indexes, number of views and number of attributes, from which 2 compound size metrics are derived

* data–points and
* functions–points.

## Measuring interface size

For measuring system and user interfaces *SoftAudit* counts 40 elementary size features such as number of parameters, number of parameter types, number of return values and number of exception conditions, and computes from them 3 compound size metrics:

* data–points,
* functions–points and
* object–points.

## Measuring test case quantity

For measuring the quantity of the test cases *SoftAudit* takes 24 elementary size counts, counts such as the count of all test cases, the count of all requirements targeted by these test cases, the count of all objects tested by the test cases and the count of all test case attributes. From these counts 3 compound size metrics are computed:

* data–points,
* functions–points and
* test–points.

## Measuring Software Complexity

Software size is determined by the number of elements – statements, attributes, methods, classes, components, widgets, etc. of which the software consists. Software complexity is, on the other hand, determined by the number of relationships between these elements. A system with many elements but few relationships is considered to have a lower complexity than a system with few elements but many relationships. The basic equation for complexity is



It is assumed that each element has at least one relationship. The lower the inverted coefficient, the less complexity the software has.

## Measuring Requirements Complexity

In measuring the complexity of requirement documents *SoftAudit* computes 6 density metrics:

* data density = number of nouns, constants and text literals relative to the number of words,
* functional density = number of actions, or verbs related to nouns, relative to the number of sentences,
* state density = number of states, or verbs related to adjectives, relative to the number of nouns, or objects,
* conditional density = number of conditional sentences and rules relative to the number of sentences,
* referential density = number of nouns, or objects, relative to the number of noun occurrences, or object references,
* test case density = number of test cases relative to the number of sentences.

The weighted average of these 6 density metrics gives the overall or median requirements complexity.

## Measuring Design Complexity

In measuring the complexity of design documents *SoftAudit* computes no less than 10 complexity metrics

* Interaction complexity = number of classes relative to the number of class associations and the number of objects relative to the number of object interactions,
* Hierarchical complexity = number of class levels relative to the number of classes,
* Data complexity = number of classes relative to the number of attributes and parameters,
* Functional complexity = number of methods relative to the number of classes,
* State complexity = number of states relative to the number of objects,
* State transition complexity = number of state transitions relative to the number of states ,
* Activity complexity = number of action nodes relative to the number of action edges,
* Use Case complexity = number of use case steps relative to the number of use cases,
* Actor interaction complexity = number of actor interactions relative to the number of use cases,
* Model complexity = number of model entities relative to the number of model relationships.

The weighted average of these 10 complexity metrics gives the overall design complexity.

## Measuring Program Complexity

In measuring the complexity of programs *SoftAudit* computes 8 complexity metrics

* data complexity = number of data references relative to the number of statements ,
* data flow complexity = number of data references relative to the number of data types
* data access complexity = number of database tables and files relative to the number of accesses to these files
* interface complexity = number of calls to foreign methods relative to the number of calls in all
* control flow complexity = number of branches – number of decisions relative to the number of branches.
* decisional complexity = weighted number of decision nodes relative to the number of statements
* branching complexity = weighted number of method calls relative to the number of statements
* language complexity = number of data types relative to number of data and the number of statements types relative to the number of statements

The weighted average of these 8 complexity metrics gives the overall program complexity.

## Measuring Database Complexity

In measuring the complexity of databases *SoftAudit* computes 6 complexity metrics

* data complexity = number of different data types and keys , relative to the sum of all data attributes stored,
* view complexity = number of views relative to the number of tables,
* access complexity = number of tables, keys and indexes relative to the number of stored attributes,
* relational complexity = number of foreign keys and relations relative to the number of tables and keys in all,
* structural complexity = number of structural elements - tables, relations and views – relative to the number of stored attributes,
* storage complexity = number of attributes stored relative to the storage capacity.

The weighted average of these 6 complexity metrics gives the overall database complexity.

## Measuring Interface Complexity

In measuring the complexity of user and system interfaces *SoftAudit* computes 6 complexity metrics

* data complexity = number of data groups relative to the number of interfaces,
* data referential complexity = number of data variables/parameters relative to the number of data references,
* data format complexity = number of titles and constants relative to the number of data variables,
* data structure complexity = number of data groups, redefinitions and arrays relative to the number of data variables/parameters,
* data usage complexity = number of operations, classes and attributes relative to the number of data variables/parameters
* language complexity = number of data types relative to the number of data variables.

The weighted average of these 6 complexity metrics gives the overall interface complexity.

## Measuring Test Case Complexity

In measuring the complexity of user test cases *SoftAudit* computes 4 complexity metrics

* Test case interdependence = number of test case relations relative to the number of test cases,
* Test case density = number of test cases to the number of requirements to be tested,
* Test case intensity = number of test cases relative to the number of data objects to be tested,
* Test case volume = number of test case attributes relative to the number of test cases.

The weighted average of these 4 complexity metrics gives the overall test case complexity.

## Measuring Software Quality

Software quality is the most difficult aspect of software to be measured because there are so many different views of what quality is. For one thing quality is relative – what is high quality for the one user may be low quality to the other. Quality can also mean different things to different people depending on their view of the software. The ISO Standard-9126 distinguishes between external and internal characteristics. External quality characteristics are features which can be observed empirically by testing such as reliability efficiency, security and usability. In principle they can be measured, at least partly, in the system source code and documentation, but it is difficult to do so and the measurement will not necessarily correspondent to how the software system really performs. In practice, systems have to be tested to obtain reliable measures of their external attributes. There are however specification, design and coding rules which when followed can contribute to better reliability, efficiency and security. The adherence to such rules is checked by *SoftAudit*.

Internal quality characteristics are features which can really only be measured over time. These are features such as maintainability, portability, reusability and interoperability. In principle, such actions have to be carried out in order to observe over time how well the system really supports them. However, there are some proven correlations between the architecture and code of a system and these economic characteristics. A system which is modularly constructed with many small interchangeable units with a minimum of dependency among them is said to be easier to maintain. A system with well defined, adaptable interfaces is said to be interoperable and a system which uses only standard operations with separated data access and user interface components is said to be portable. Finally a system which consists of cohesive parts which are loosely coupled with one another is said to be reusable since it is possible to extract individual blocks of code without affecting the others. More than 30 years of research has gone into demonstrating the relationship between measurable internal characteristics of software systems and the costs of maintaining, migrating, reusing and integrating them. So the owners of IT systems can only believe what the researchers tell them or not. If not, they have to experiment for themselves.

In considering source code, requirement and design documentation there are certain features which lend themselves to being measured. What can be measured depends on the type of software artifacts. Interfaces and databases have other qualities than programs. Object–oriented programs also differ in their qualities from procedural programs. Requirement documents have their own special qualities as do design models.

## Requirement Quality Measurement

Requirement quality is determined by the degree of formalism, the way the document is structured and by the completeness and consistency of the content. In regards to the quality of requirements *SoftAudit* computes 6 quality metrics. These are:

* Completeness = number of missing object and usecase attributes relative to the number of required attributes ,
* Consistency = number of references to undefined entities relative to the number of references ,
* Stability = number of changed requirements relative to the sum of all requirements,
* Changeability = number of requirement titles (sections, requirements, use cases, objects) relative to the number of sentences ,
* Testability = number of user interfaces, panels, reports and use cases relative to the number of test cases
* Conformity = weighted deficiencies relative to the number of sentences

The weighted average of these 6 quality metrics gives the overall requirement quality.

## Design Quality Measurement

Design quality is similar to program quality only here the quality is that of the architectural framework, i.e. the structure of the system as a whole. It depends on certain features of the architecture. To measure the quality of the design *SoftAudit* computes altogether 10 design quality characteristics. These are:

* Coupling = number of classes and objects relative to the number of associations and interactions,
* Cohesion = number of class methods relative to the number of class attributes ,
* Modularity = number of methods relative to the number of classes ,
* Portability = number of components relative to the number of classes and interfaces ,
* Reusability = number of generalizations, associations and interactions relative to the number of classes and methods
* Testability = number of test cases relative to the number of classes, interfaces and use cases ,
* Conformance = number of deficient design entities relative to the number of entities,
* Completeness = number of existing design diagrams relative to number of planned design diagrams ,
* Consistency = number of design entities defined relative to the number of design entities referenced,
* Compliance = number of use cases designed relative to the number of use cases required.

The weighted average of these 10 quality metrics gives the overall design quality.

## Program Quality Measurement

Internal program quality is determined many factors. In regards to the quality of programs *SoftAudit* measures 8 quality characteristics. These are:

* Modularity = Mean of class cohesion, class coupling and class size
* Portability
* flexibility
* testability
* convertibility
* reusability
* interoperability
* conformity.

*(See Figure 3:Auditing and measuring Code Artifacts)*



## Database Quality Measurement

The qualities of data have to do with the independence of the data from the applications, the variety of access paths, the adaptability of the data structures without unwanted side effects, the efficiency of how the data is stored, the security of the data, i.e. to what extend the data is coded and the conformity with prevailing company standards, in particular concerning the naming conventions. To this end, *SoftAudit* computes 6 data quality metrics

* data independence = number of views and indexes relative to the number of tables or record types
* data accessibility = number of keys, indexes and selects relative to the number of tables,
* data flexibility = number of attributes and keys relative to the number of tables,
* data testability = number of test cases relative to the number of data attributes stored,
* data storage efficiency = number of attributes stored relative to the size of the storage in bytes,
* data conformity = number of schema deficiencies relative to the number of schema lines.

The weighted average of the 6 quality metrics gives the overall database complexity.

## Interface Quality Measurement

The quality of user and system interfaces has to do with the usability, the adaptability, the reusability, the testability, the portability and the conformance of the interface with prevailing standards. Of course, the final usability of a user interface can only be fully measured by using or misusing it. However, there are features of the interface descriptions which indicate usability and these features can be measured statically. Therefore, *SoftAudit* also measures 6 quality characteristics of user and system interfaces. These are

* Modularity = number of data variables/parameters relative to the number of interfaces,
* Adaptability = number of interfaces relative to the number of data groups
* Reusability = number of constants and titles relative to the number of data variables,
* Testability = number of test cases relative to the number of interfaces and data variables
* Portability = number of statement types relative to the number of statements,
* Conformity = weighted deficiencies relative to the number of code lines.

The weighted average of the 6 quality metrics gives the overall interface complexity.

## Test Case Quality Measurement

The quality of the test cases is determined by two factors. One factor is their degree of completeness and consistency. All of the required test case attributes should be present and they should be consistent with one another. The other factor is that they cover the requirements. For every requirement, use case and data object there should be at least one test case. To check the degree to which these two factors are satisfied S*oftAudit* computes 4 test quality metrics

* Testability = number of test cases relative to the number of test objects
* Test coverage = degree to which the functional and non-functional requirements are covered by the test cases,
* Test completeness = degree to which all of the mandatory test cases are defined,
* Test reusability = degree to which the test cases are automated.

The weighted average of these 4 quality metrics gives the overall test case quality.

## Aggregating the Measurements

The metrics of all six semantic levels – requirements, design, program, database, interface and test case - of all three types of measurement – quantity, complexity and quality - are aggregated:

* for each component
* and each subsystem
* and each product.

A software product may contain one or more subsystems. Thus, the product metrics are an aggregation of all the system metrics. In the case of quantity metrics it is their sum. In the case of quality and complexity metrics it is their average.

The system metrics are an aggregation of the component metrics. The component sizes are summed up; the component qualities and complexities are averaged.

The components are the lowest level of measurement. Components are the requirement documents and the design subsystems. At the code level, they are the programs in procedural systems, the modules in object–oriented systems, the database schemas at the database level and the interface schemas at the interface level. The modules in object–oriented systems may contain any number of classes, just as programs in procedural systems may contain any number of procedures. There are no metrics at the class, procedure or database table level. The counts of these elementary software artifacts go into the metrics at the component, database or interface level.

For more information the reader is referred to the SoftAudit tool documentation. (Tools\Measure\SoftAudit\Docus\SoftAudit.pdf)

# Collecting Productivity Data with SoftProd

In order to project the number of change requests and error reports and to calculate the costs of software evolution projects, it is necessary to have two basic types of measurement data:

* Product data and
* Process data

The product data is delivered by the *SoftAudit* tools. It consists of all measures of the size, complexity and quality of the product. The process data is delivered partly by the *SoftTest* tools which record test coverage and error frequency. What is missing is the data on pending error reports and change requests as well as data on the expended effort. A software product manager must know how many errors were reported and how many change requests were submitted in the past time period. Furthermore, he should know how much effort was spent on the evolution of a product in that time period. So we are talking here about three kinds of process measurement data required for planning software evolution:

* Error report data
* Change request data and
* Expended effort data.

By joining these three kinds of measurement data, one comes up with evolution productivity data, which is the goal of the tool *SoftProd*. In this respect *SoftProd* is a prerequisite for using *SoftCalc* to estimate software evolution projects. *(see Figure 4: Measuring the Software Evolution Process)*



## 2.1 Obtaining Data from Electronic Forms

Error reports and change requests are normally made with electronic forms. These forms contain blocks of data identified by their titles. The data are then stored either in text files, XML files or in tables. In the text files, data lines are preceded by title lines identifying the semantic meaning of the data which follows. In XML files the data contents are identified by their tags. There can also be attributes describing the data. In tables the data entries are stored in fixed columns of the table, whereby each column has a header to identify the content of the table. The tables can be exported in the form of comma separated value files. Therefore, to obtain data from error reports and change requests, the type of electronic form used must be given as a parameter. There are three possible source form types:

* Marked up text files
* Tagged xml files
* Csv tables

The tool *FormScan* is intended to select data from electronic forms. The user of FormScan must submit the form type – txt, xml or csv – and the keywords of the data attributes he wants to extract from the form. The titles of the data in the form are connected by the keywords used by the tool in a keyword table. It doesn’t matter how the data is titled in the electronic. It can be in German, French, Hungarian or any other language, only the same title must be used consistently in all forms and the user must assign this title to a system key word. This keyword table is of course stored and reused for subsequent form processing. The user then identifies the directory where the electronic forms are stored. When they are displayed, he can select which of the forms he wants to process and starts the background job to process them. The two outputs of the job – an xml metric file for *SoftEval* and *SoftCalc* as well as a summary report – are placed in an output directory from whence they can be printed out or further processed. The xml metric file is available here to be read in by the *SoftEval* and *SoftCalc* tools.

## 2.1.1 Processing Error Reports

Error reports normally contain all the information required by the developer in order to locate and correct the error. Most of this information is significant only to the persons involved. For the purpose of evolution management only the following subset of attributes is needed:

* the product affected
* the system affected
* the artifact affected
* the type of error
* the source of error
* the severity of the error
* the estimated cost of the error
* the reporting date
* the closure date
* the reporting source – user, tester or developer.

It is up to the tool *FormScan* to identify these attributes and to extract them from the form.

The product and system names are labels assigned to software products and their parts. The artifact is the label of the document, database, interface or code component where the error was found. The types of errors are logic errors, data errors, user errors, integration errors, etc. The source of error is, if known, where the error originated – in the requirements, in the design, in the coding or in the build. The severity of the error is taken from the IEEE Standard 1044 on error classification. It can be critical, severe, major, minor or only disturbing. The cost of an error is the maximum loss in working hours which can occur as a result of that error. With this data it is possible to make a defect analysis and to keep the product manager informed on the status of the product. The frequency of defects combined with the error severities and the potential defect costs are important decision making parameters.

## 2.1.2 Processing Change Requests

Change requests are usually less structured than error reports. Besides some fixed attributes they contain a prose text on what should be done to alter or enhance the software. The prose text can be handled in the same way as requirements are handled. It should be extracted into a separate text document, marked up and analyzed by the *SoftAudit* text analyzer. What is of interest here are those fixed attributes which identify what feature of the existing software is to be altered or added, what the requester is willing to pay for the change, what is the maximum time allowed and what is the expected benefit. The actual cost of the change can only be determined after the impact analysis. For the purpose of evolution management the following subset of attributes is needed:

* the product affected
* the system affected
* the component affected
* the type of change
* the priority of change
* the maximum cost of the change
* the added value of the change
* the date of request
* the latest date of delivery
* the actual date of delivery
* the requesting source – user, tester, developer or manager.

It is up to the tool FormScan to identify these attributes and to extract them from the form.

The product and system names are labels assigned to software products and their parts. The component affected could be a use case, an interface or a database. It will serve as a starting point for the subsequent impact analysis. The type of change is whether it is adaptive, perfective or enhancive. The priority of a change can be low, medium or high. The maximum cost is the maximum amount the requestor is willing to pay. The added value of the change is the projected percent that the value of the system will increase by making this change. This information suffices to start an impact analysis and to make a feasibility study. If the change project will cost more than what the requester is willing to pay or if it requires longer than the requestor is willing to wait, then there is no sense in making the change. The change should also bring some added value relative to the costs.

## 2.1.3 Processing Time Sheets

The time, spent by persons working on projects, is usually recorded in some kind of time management system. From the time management system XML or CSV files can be produced. These can then be analyzed to pick out the data required for planning. The goal here is to record the effort spent on past projects or releases for use in calculating future evolution projects or product releases. For that reason it is important that the time worked be assigned to particular tasks, e.g. to error corrections, to changes, to integration or to further development. Thus, of interest here are only three simple attributes:

* the product worked on
* the system worked on
* the maintenance task (error report or change request) worked on
* the type of maintenance work (corrective, adaptive, perfective or enhancive)
* the hours worked
* the month or week in which the work was done.

These attributes will be picked out of the stored time sheets and packed into an XML file for importing them to the measurement tools *SoftEval* and *SoftCalc*.

## 2.2 Compiling Productivity Data

The quality of an evolution process can only be judged in terms of certain process metrics. Just as the product metrics allow us to compare software products with one another and to some benchmark product, process metrics allow us to compare software processes with one another and to a given process benchmark. The four key process indicators - KPIs – are the evolution tasks accomplished, the weighted defects caused by those tasks, the time required to finish those tasks and the effort spent working on those tasks. The goal is to perform as many evolution tasks as possible within the shortest time possible with the minimum of effort and the lowest possible defect rate. The product measurements of size, complexity, quality, conformance and coverage may be interesting to follow, but they are secondary to the process measurements for productivity and reliability. These are the primary indicators of good performance. IT departments should accomplish more tasks in less time with less cost and fewer errors. That is the bottom line of software productivity.

To support the management of evolution, the tool SoftProd maintains a small database of key performance indicators. Here the reported errors, the requested tasks, the completed tasks and the effort expended are recorded. The user may at any time view this data in tabular format and edit it. There is a window to each data table:

* the error report table
* the change request table
* the maintenance task table.

The error report table contains a row for each reported error with a column for each error report attribute. The change request table contains a row for each requested change with a column for each change attribute. The maintenance task contains a row for each maintenance task with a column for each task attribute, i.e. time elapsed and effort expended. It will certainly be necessary to edit this data prior to exporting it over to *SoftEval* for evaluation and to *SoftCalc* for calculation.

## 2.3 Producing Productivity Reports

From the productivity data base three productivity reports can be produced on demand.

* A summary report of all error reports by product and system
* A summary report of all open error reports
* A summary report of all change requests by product and system
* A summary report of all pending change requests
* A summary report of the effort expended by task (error correction or change).

In addition to these reports the tool *SoftProd* generates three XML export files for transferring data to SoftEval and SoftCalc:

* A defect metric file
* A change metric file and
* A productivity file.

## 2.3.1 Defect Metric File

The defect metric file contains 8 defect counts plus a test effectiveness ratio for each subsystem. The defect counts are:

* tester reported defects
* customer reported defects
* total reported defects.
* fatal defects
* major defects
* medium defects
* minor defects
* total defects.

The test effectiveness ratio is the coefficient of tester reported defects divided by total defects. There is a record for each subsystem.

## 2.3.2 Change Metric File

The change metric file is similar to the defect metric file, only it has 7 change request counts plus a volatility ratio. The 7 change request counts are:

* major changes
* average changes
* minor changes
* total changes
* self requested changes
* customer requested changes
* total requested changes

The volatility ratio is the coefficient of the customer requested changes divided by the total number of change requests. Here too there is a record for every subsystem.

## 2.3.3 Time Metric File

The time metric file contains a record for each task with the attributes

* project-id
* deliverable
* activity
* time elapsed
* hours worked

The hours worked and the time elapsed are numeric values. The other attributes are all coded identifiers.

# Evaluating Software Products with SoftEval

Once a software system has been put into operation, it must evolve in order to remain useful. Initial errors must be corrected, changes must be made and functions added. According to Lehman’s laws of software evolution, as a system grows over time, its complexity will increase whereas its quality will sink. Eventually the complexity will be so great and the quality so low that the system is no longer maintainable. It will have reached the saturation point. The five laws of software evolution:

* The law of continual change,
* The law of increasing size and complexity,
* The law of decreasing quality,
* The law of diminishing productivity, and
* The law of restricted growth

were proclaimed by Belady and Lehman as early as 1975 and have been confirmed over the years. The fact is that software ages and deteriorates thru change. Eventually it becomes petrified.

To prevent a software system from becoming petrified und unchangeable, the software should be periodically evaluated and if necessary reengineered. Software Reengineering is the means by which the quality of a system is increased and the complexity decreased. Software Evaluation is the method through which one can determine when a system needs to be reengineered. Evaluations are also conducted to determine whether a software system should be accepted or not when an organization is taking over a system from another organization or when it is being delivered from an outsourcing partner. The goal is in all cases to determine the state of the product. Since software systems are so large and so complex tools are needed to accomplish this goal. That is the purpose of the software evaluation tool - SoftEval.

*SoftEval* collects metric data in a metric database from various sources, from the requirements analysis, from the design, from the source analysis, from the test analysis, from the error reporting system and from the change management system. The main providers of metric data are the tools *SoftAudit, SoftProd* and *SoftTest*. *SoftAudit* has been described above. It statically analyzes requirement documents, design models, source codes, databases, interfaces, and test cases to obtain the metrics required to make an evaluation. SoftProd ist described later in this documentation. It collects data on the productivity of projects and the errors rates of products. *SoftTest* is described in a separate document. In dynamically analyzing a software system, it collects measurement data on the executed test cases, the test coverage, the test data usage and the error frequency. This measurement data is passed on to *SoftEval* in the form of XML files. *SoftEval* then filters and aggregates the data before storing it in the metric data base. The creation of a metric data base, to be used as a general source of numeric information, can be seen as a secondary goal, as a means to an end. The primary goal is to evaluate the software product. However to achieve this goal it is first necessary to gather and store data about the product.

Once the data has been collected in a database, *SoftEval* can produce a series of reports on the size, complexity, quality, stability and volatility of a software product giving the product manager the information needed to make decisions as to what to do with the product, i.e. when to reengineer it, when to freeze it and when to replace it. *SoftEval* supports the decision making process by providing reports and views on the metric data base of each product version, thus allowing the comparison of versions. Thus, it not only maintains a metric database, but it also uses that database to supply numerical information. The four major functions of the *SoftEval* tool are:

* to collect and store metric data in a metric database,
* to edit the metric data,
* to produce reports from the metric database, and
* to create a manager’s dashboard for monitoring the state of a product. *(See Figure 5: Evaluating Software Products )*



## 3.1 Collecting and Storing Metric Data

Metric data is collected by *SoftEval* from 11 different sources:

* from the requirement documents (Ger- and EngAudit),
* from the design models (UMLAudit),
* from the source code (xxxAudit),
* from the database descriptions (yyyAudit),
* from the data interface definitions (zzzAudit),
* from the test cases (TC-Audit)
* from the test data protocols (DataTest),
* from the test execution logs (TestDoc),
* from the error reports (SoftProd)
* from the change requests (SoftProd), and
* from the time sheets (SoftProd).

## 3.1.1 Storing Requirements Metrics

Requirements metrics are generated by the tool *TextAudit* when it analyses the German and English language requirement texts. It measures the size, complexity and quality of the requirements. The metrics are exported in the form of an XML file with the name of the required system. *SoftEval* imports the file, parses it and stores the requirement metrics in the requirement metric table of the metric database under the name of the product and the system.

## 3.1.2 Storing Design Metrics

Design metrics are generated by the tool *UML*Audit when it analyses the UML design models counting the model elements – entities and relationships. In this way it measures the size, complexity and quality of the design model. The metrics are exported in the form of an XML file with the name of the proposed system. *SoftEval* imports the file, parses it and stores the design metrics in the design metric table of the metric database under the name of the product and the system.

## 3.1.3 Storing Source Code Metrics

Source code metrics are generated by the *CodeAudit* tools when they analyze the various source code languages. The size, complexity and quality of the code are measured according to the coding paradigm, either procedural or object-oriented. The metrics are exported in the form of XML files aggregated at the subsystem level. *SoftEval* imports the files, parses them and stores the interface metrics in the code metric table of the metric database under the name of the product and that particular subsystem.

## 3.1.4 Storing Data Metrics

Data metrics are generated by the *DataAudit* tools when they analyze the coded database schemas. They measure the size, complexity and quality of the databases depending on whether they are hierarchical, networked or relational. The metrics are exported in the form of XML files with the name of the database analyzed to identify them. *SoftEval* imports the files, parses them and stores the data metrics in the database metric table of the metric database under the name of the product and the database.

## 3.1.5 Storing Interface Metrics

Interface metrics are generated by the *InterAudit* tools when they analyze the various interface definitions. They measure the size, complexity and quality of the interfaces based on the interface type, whether API, IDL, XML or WSDL. These metrics are then exported in the form of XML files bearing the name of the interface analyzed. *SoftEval* imports the files, parses the contents and stores the interface metrics in the interface metric table of the metric database under the name of the product and the system to which the interfaces belong.

## 3.1.6 Storing Test Case Metrics

Test case metrics are generated by the tool *TC-Audit* when it analyses test cases. The test case tables are processed to measures the quantity, complexity and quality of the test cases. These metrics are then exported in the form of XML files with the name of the system for which the test cases are intended. SoftEval imports the files, parses them and stores the test case metrics in the test case metric table of the metric database under the name of the product and the system under test.

## 3.1.7 Storing Test Data Metrics

Test data metrics are generated by the data testing tool – *DataTest. DataTest* generates correctness and data coverage metrics when it analyses the data results of a test. The result of the data validation measurement is sent as an XML File for each data object tested to *SoftEval*. *SoftEval* imports the files, parses them and joins the correctness and coverage metrics of each file, database and interface tested into a single row per system in the test data metric table of the metric database under the name of the product and the system.

## 3.1.8 Storing Test Object Metrics

Test object metrics are generated by the test documentation tool – *TestDocu*. *TestDocu* generates reliability and code coverage metrics when it analyses the execution of a system test. These metrics are merged into a XML file per system under test and passed on to *SoftEval*. *SoftEval* imports the files, parses them and stores the correctness, reliability and coverage metrics in the test case metric table of the metric database under the name of the product and the system.

## 3.1.9 Storing Defect Metrics

Defect metrics are extracted by the tool *SoftProd* from the problem reporting system by scanning the error reports and extracting key defect data. The key defect data is the number of defects by defect category, by time period, by source of defect, i.e. which component, and a source of the problem report, e.g. developer, tester or customer. These metrics are exported in the form of an XML file with the name of the system containing the defects. *SoftEval* imports the file, parses it and stores the defect metrics in the defect metric table of the metric database under the name of the product and the system.

## 3.1.10 Storing Change Metrics

Change metrics are extracted by the tool *SoftProd* from the change management system by scanning the change requests and extracting key change data. The key change data is the number of change requests by change category, by time period, by target of change, i.e. which use case and by source of the change request, i.e. which customer. These metrics are exported in the form of an XML file with the name of the system to which the changes apply. *SoftEval* imports the file, parses it and stores the change metrics in the change metric table of the metric database under the name of the product and the system.

## 3.1.11 Storing Time Metrics

Work time data is collected by the tool *SoftProd* by scanning the time sheets and extracting key time data. The key time data is the number of hours worked by worker, by time period, by project and by deliverable, i.e. which document, component, test, etc. These metrics are joined together and exported in the form of an XML file with the name of the project for which the work was done. *SoftEval* imports the file, parses it and stores the time metrics in the time metric table of the metric database under the names of the project and the product.

## 3.2 Editing Metric Data

The editing of the metric data is done by displaying the metric table and allowing the user to overwrite selected lines and attributes. There is a separate window for each of the following 11 tables:

* Requirement Metric Table
* Design Metric Table
* Code Metric Table
* Data Metric Table
* Interface Metric Table
* Test Case Metric Table
* Test Data Metric Table
* Test Object Metric Table
* Defect Metric Table
* Change Metric Table
* Time Metric Table

in which the table contents are displayed. The user needs only to select a row and to overwrite the column values.

## 3.2.1 Requirement Metric Table

The requirement metric table contains 9 subgroups of attributes:

* Text Quantities
* Interface Quantities
* Data Quantities
* Functional Quantities
* Use Case Quantities
* Deficiency Quantities
* Size metrics
* Complexity metrics
* Quality metrics.

There is a row for each requirement document plus a row for each subsystem and a row for the product as a whole. Each row contains the name of the document plus a column for each metric. The names of the metrics are given in the title line.

## 3.2.2 Design Metric Table

The design metric table contains 5 subgroups of attributes:

* Design quantities
* Design sizes
* Design deficiency counts
* Design complexity metrics
* Design quality metrics

Here too, there is a row for each design document plus a row for each subsystem and a row for the product as a whole. Each row contains the name of the design artifact plus a column for each metric.

## 3.2.3 Code Metric Table

The code metric table contains 8 subgroups of attributes:

* Source text quantities
* Code structure quantities
* Data structure quantities
* Procedural quantities
* Code sizes
* Code deficiency counts
* Code complexity metrics
* Code quality metrics

There is a row for each subsystem as well as a row for the product as a whole. In the first column is the name of the program or interface followed by a column for each code metric. The quantities and sizes are integers, the complexities and qualities are ratios.

## 3.2.4 Data Metric Table

The data metric table contains 7 subgroups of attributes:

* Source text quantities
* Data structure quantities
* Data type counts
* Data sizes
* Data deficiency counts
* Data complexity metrics
* Data quality metrics

There is a row for each database plus a row for the sum of all databases audited. The first column is the name of the database, followed by a column for each data metric. The quantities and sizes are integers, the complexities and qualities are ratios.

## 3.2.5 Interface Metric Table

The interface metric table contains 7 subgroups of attributes:

* Source text quantities
* Interface structure quantities
* Parameter type counts
* Interface sizes
* Interface deficiency counts
* Interface complexity metrics
* Interface quality metrics

There is a row for each interface plus a row for the sum of all interfaces of a product. The first column is the name of the interface, followed by a column for each interface metric. The quantities and sizes are integers, the complexities and qualities are ratios.

## 3.2.6 Test Case Metric Table

The test case metric table contains 7 subgroups of attributes:

* Test case attribute counts
* Test case reference counts
* Test data type counts
* Test case deficiency counts
* Test case sizes
* Test case complexity metrics
* Test case quality metrics.

There is a row for each subsystem under test plus a row for the product as a whole. The first column of each row identifies the subsystem and product. It is followed by a row for each test case metric.

## 3.2.7 Test Data Metric Table

The test data metric table contains the data metrics produced by the validation of the databases and system interfaces in the system test. The tool *DataTest* delivers three types of metrics:

* Data quantity metrics
* Data coverage metrics
* Data discrepancy counts.

They are stored in a relational table with a row for each interface, file and database table validated plus a row for the product as a whole. The first column identifies the test object. The subsequent columns contain the counts and ratios.

## 3.2.8 Test Object Metric Table

The test object metric table is created from the metrics provided by the tool *TestDoc* which monitors the system test. Four types of test metrics are stored here for each subsystem tested:

* Test quantity metrics
* Test coverage metrics
* Defect counts
* Test quality metrics.

In this table there is a row for each subsystem tested plus a row for the product as a whole. The first column identifies the test object. The subsequent columns contain the counts and ratios.

## 3.2.9 Defect Metric Table

The defect metric table contains 8 defect counts plus a test effectiveness ratio for each subsystem. The defect counts are:

* tester reported defects
* customer reported defects
* total reported defects.
* fatal defects
* major defects
* medium defects
* minor defects
* total defects.

The test effectiveness ratio is the coefficient of tester reported defects divided by total defects. There is a row for each subsystem and for the product as a whole.

## 3.2.10 Change Metric Table

The change metric table is similar to the defect metric table in that it has 7 change request counts plus a volatility ratio. The 7 change request counts are:

* major changes
* average changes
* minor changes
* total changes
* self requested changes
* customer requested changes
* total requested changes

The volatility ratio is the coefficient of the customer requested changes divided by the total number of change requests. Here too there is a row for every subsystem plus a row for the product as a whole.

## 3.2.11 Time Metric Table

The time metric table contains a row for each project with the attributes

* project-id
* deliverable
* activity
* time elapsed in days
* hours worked

Only the hours worked is a numeric value. The other attributes are all coded identifiers.

## 3.3 Producing Metric Reports

SoftEval offers the user the option for generating one or more of the following metric reports:

* requirement metric table
* requirement quantity ranking
* requirement complexity ranking
* requirement conformity ranking
* requirement quality ranking
* requirement comparison report
* design metric table
* design quantity ranking
* design complexity ranking
* design conformity ranking
* design quality ranking
* design comparison report
* code metric table
* code quantity ranking
* code complexity ranking
* code conformity ranking
* code quality ranking
* code comparison report
* database metric table
* database quantity ranking
* database complexity ranking
* database conformity ranking
* database quality ranking
* database comparison report
* testcase metric table
* testcase quantity ranking
* testcase complexity ranking
* testcase conformity ranking
* testcase quality ranking
* testcase comparison report
* test metric table
* test quantity ranking
* test complexity ranking
* test conformity ranking
* test quality ranking
* test comparison report
* defect metric table
* defect ranking
* defect comparison report
* change metric table
* change ranking
* change comparison report
* time metric table
* time ranking
* time comparison report
* project metric report
* project ranking
* project comparison report
* product metric report
* product ranking
* product comparison report.

## 3.3.1 Metric Tables

The metric tables produced are Excel tables for each product and project in which the rows are the subsystem of the products or in the case of projects and time the activities. The first column contains the name of the product or project. The second column contains the subsystem or activity name. The third column gives the source from which the metrics were derived. The fourth column gives the date of the measurement. The remaining columns contain the counts of that subsystem or activity. The names of the counts are given in the header line. The last row contains the sums of the columns. This table is intended to give the user an overview of the software sizes, the defect and change counts and the times recorded.

## 3.3.2 Ranking Reports

The ranking reports for the product elements are of four kinds

* ranking by size,
* ranking by complexity,
* ranking by conformity, and
* ranking by quality.

The elements – documents, subsystems, and tests – are listed out in descending order from the highest ranked to the lowest. At the end the average and the medium rankings are given. Defects and change requests are only ranked by the number per subsystem, i.e. by quantity. Time and Projects are ranked by effort.

Time is ranked by the reporting person. Those which report the most hours worked are at the top whereas those with the least hours worked are at the bottom.

Projects are also ranked by time. Those projects with the most hours booked are at the top of the list, those with the least hours are at the bottom. The average and the median number of hours worked is computed for both the time and the project ranking.

## 3.3.3 Comparison Reports

The comparison reports compare the metrics of one product or project with the metrics of another and report the percent difference. It can also be a comparison between two different versions of the same product. In the first column the names of the metrics are listed, in the second column, the values for the old or base product / project are listed, in the third column the values of the new or target product / project are listed, and in the fourth column the degree of difference in +/- percent is given. With this report the user can see how products, product versions and projects differ and to what degree they differ. Since, in software everything is relative; this report is the ultimate tool in assessing software products and projects. The prerequisite is of course that the metrics all come from the same sources, e.g. *SoftAudit, SoftTest and SoftProd.* This is a good reason for using an integrated measurement workbench, since this is the only way to ensure comparability.

## Providing Graphic Displays of Metric Data

*SoftEval* has the additional function of providing graphic displays of the metric data contained in the metric database. This can be referred to as “cockpit” function. Five different types of graphics are provided:

* Bar Charts
* Pie Charts
* Fishbone Diagrams
* Distribution Curves
* Management Dashboard

## 3.4.1 Bar Charts

The histograms show the relative percentage of total size that the measured subsystems have compared to the whole. The size varies. For requirements, designs, programs, interfaces and databases it can be any of the common size metrics such as lines, statements, function-points, object-points, test-points, etc. For the test it is the number of test cases and defects. For the defects it is the number of defects per system. For the changes it is the number of change requests per system. For productivity it is the effort per project. The bars represent the individual subsystems.

## 3.4.2 Pie Charts

The pie charts show how the whole is distributed among the parts. The whole can be the total size or the total number of deficiencies, changes and defects. These charts give the user an overview of the proportions that each system has of the entire software product.

## 3.4.3 Fishbone Diagrams

The fishbone or kiviat diagram describes to what degree a set of measurable goals are fulfilled. In *SoftEval* two such diagrams can be produced, one for each software artifact type per subsystem. The one diagram displays the degree to which the complexities vary on a rational scale of 0 to 1, the goal being to be as close to zero as possible. The spokes of this diagram are the individual complexity measures of each software artifact type. The other diagram displays the degree to which the qualities vary on a rational scale of 0 to 1, the goal being to be as close to one as possible. The spokes of this diagram are the individual quality metrics. There can be a kiviat diagram for each artifact type, i.e. requirements, design, code, database, interface and test cases. Of course, they cannot be mixed.

## 3.4.4 Distribution Curves

The distribution curve is used to depict the relation between two discrete metrics. In *SoftEval* it is used to depict the relation between system size and the number of defects as well as to defect the relation between system size and effort. The vertical axis is the system size. The horizontal axis is the number of defects or the number of person months. The first curve depicts the error density, the second depicts the productivity.

## 3.5.5 Manager’s Dashboard

The fifth graphic which *SoftEval* produces is a graphic display of the manager’s dashboard. This is a combination of graphic indicators or gauges. Here there is a single surface with a total of eight gauges. The upper left gauge displays the complexity rate. The upper left gauge displays the quality rating. The upper middle gauge displays the test coverage rate. The middle left gauge displays the project productivity in points per person month. The middle right gauge displays the error density in errors per thousand instructions. The lower left gauge displays the change rate, i.e. system volatility. The lower gauge displays the deficiency rate in deficiencies per statement. The lower middle gauge depicts the degree to which the requirements are fulfilled. Each gauge has a green, a red and a yellow area. If the indicator is the green area, then the state is better than expected. If it is in the yellow area then it is in an acceptable state. If the indicator falls into the red area, it is a sign that the system is in trouble. With the help of the dashboard, the manager is able to perceive in one glance what the status of a software system is relative to what it should be. It is then up to him, to instigate the appropriate measures to bring the system back on course. *(see Figure 6: Visualizing the state of an Evolution Project)*



# Calculating Software Projects with SoftCalc

Accurate cost estimation is essential for successful software projects. If the costs and the deadlines are not calculated well, the business case for the project becomes worthless. The whole cost/benefit analysis is thrown off. That is why in reality every software project is a fixed price project. To justify the costs of a project the costs must be known before the project is started. This is the purpose of software project calculation.

To make an accurate estimation of a project it is necessary to have certain information about the product to be produced and the project to be carried out. This information is referred to as the five core metrics:

* the size & complexity of the product, i.e. the functional requirements,
* the quality of the product, i.e. the non-functional requirements,
* the productivity of the project, i.e. how many size units can be produced per time unit,
* the time needed to finish the project, i.e. the expected deadline,
* the effort required to make the project, i.e. the costs.

The relation of these five parameters to one another was coined already by Sneed in 1987 as the Devils Quadrat. 15 years later Putman and Myers declared these to be the cornerstones of software cost estimation. In fact, all algorithmic cost estimation is based on this simple quadratic equation.

effort = {(size·quality) / productivity} *time*

One of the parameters – productivity – is given. Two of the parameters may be set and the other two are dependent on the relationship of the first two to the constant. Thus, by setting size and quality, effort is derived from the relationship between size and quality on the one side and productivity on the other

effort = ( size·quality )

productivity

By setting effort, size and quality will be derived from the relationship between effort on the one side and productivity on the other.

size·quality = ( effort )

productivity

Time is a function of effort and productivity adjusted by an exponent for work distribution.

Time = ( effort ) *Distribution Exponent*

productivity

Size and quality can be taken from the product description or from the product itself. Time and effort can be dictated by the customer. The key factor is the productivity factor, which can only be derived from empirical studies of previous projects.

Productivity is known to vary significantly in the field of software. Studies, in particular those of Sachman, Weinberg and Maxwell, have noted a variance of 1 to 10 in the productivity of individuals and teams. So how is a software shop going to be able to predict its productivity? The answer to that is through observation of its own project experience.

Only those who collect and record their experiences can learn from them, the others are doomed to repeating their mistakes. That is the Darwanian law of software evolution. Those who learn from past projects and carry this experience over into their current projects have a much greater chance of surviving than those who have accumulated no experience or if they have, discarded it.

For this reason, the key ingredient of the *SoftCalc* system is the project productivity database. All cost estimations, no matter with that method, are projections of past experience, i.e. the productivity of past projects is projected to predict the productivity of the next project. Once, the productivity has been established it is only a question of matching that productivity with the size, complexity and quality of the planned product and adjusting it with relevant project influence factors. The productivity table, the product metrics and the project influence factors are, therefore, the main inputs to the *SoftCalc* tool. The outputs are estimated time and effort of the planned project. The functions of *SoftCalc* are

* to define a software project
* to import a productivity table
* to import system metrics
* to adapt system metrics
* to prescribe system quality
* to set project influence factors
* to rate project resource factors
* to analyze project risk factors
* to calculate project costs and time
* to project error rates
* to predict maintenance costs. *(See Figure 7: Calculating the Costs of Software Evolution Projects)*



## 4.1 Functions of the SoftCalc tool

## 4.1.1 Defining a Software Project

The first step in preparing a software estimation is to define the project. This is done with *SoftCalc* by assigning a project and a product name, determining the project type and technology, and by setting the upper bounds of the project costs, the project duration and the project change rate. This information is required for creating an initial project definition.

## 4.1.2 Importing a Productivity Table

The second step in preparing software estimation is to import a productivity table. The estimating organization should make a study of past projects in order to collect data on the time and effort required to finish those projects as well as on the sizes of the products delivered by those projects. This data can be collected into a standard Excel table and then converted to an XML file to be loaded into the calculation database.

Without having any own project experience data, the estimating organization will have no other choice than to adapt one from another organization or to use the standard productivity table offered together with the *SoftCalc* tool. The imported data must be in the form of an XML file with fixed attributes.

## 4.1.3 Importing System Metrics

The third step in preparing software estimation is to import the system size, complexity and quality metrics. These will come from either the requirements analysis tool, the testing tools or from SoftAudit.

The requirements analysis tool – *TextAnal* – creates a requirements metric export file in Xml format containing the:

* use cases,
* data objects,
* system interfaces, and
* initial test cases.

This file is imported by SoftCalc to initialize the use case, object, interface and test case tables.

The test tools – *TestCase Analyzer, Testdocumentor and DataTester* – create three exports, one from each of the test tools.

* final test cases,
* data and component coverage, and
* test statistics.

The test statistics include various coverage rates, defect rates and test metrics such as remaining error probability, test efficiency and test effectiveness. These files are imported by *SoftCalc* to finalize the test case table and to fill out the product data.

The tool *SoftAudit* provides export files from the source analysis of code components databases and system interfaces. These are the

* code entity file
* data entity file and
* interface entity file.

The code entity file contains not only the components, but also the data objects and the system or use interfaces embedded in the programs. The data entity file contains the object types actually declared within the database schemas. The interface entity file contains all of the system and user interfaces implemented. These files are imported by *SoftCalc* to create the code component table and to update the object and interface tables. Importing this metric data from the text, test and source analysis tools is more accurate than when the user submits the data manually and is, of course, much faster.

## 4.1.4 Adapting system metrics

It is not necessary to import product data in order to make estimations. All data can be submitted manually. Thus the fourth step in preparing software estimation is to insert or update the system size metrics. *SoftCalc* has a table for each of the major system entities

* Processes or Use cases,
* Data objects,
* System interfaces,
* Components,
* Test Cases.

These tables can be displayed to the user so that he can edit them. Entities can be added, changed or deleted. Each entity has four size measurements and a change rate. The size measurements are used to calculate data–points, function–points, object–points, usecase- points, test-points and deliverable statements. The change rate denotes the percentage of change required for each entity. Developing an entity from scratch implies 100% change, whereas changing an existing entity may only involve a 10% change. The sizes of the system entities are adjusted by their change rate which is the inverse of the reusability rate.

## 4.1.5 Prescribing System Quality

Size alone is not enough to determine the effort required to produce a product. There is also the quality dimension. Some methods such as the Function–Point method try to avoid this issue. Quality means different thing to different people. It is difficult to define and difficult to measure. Nevertheless it is essential to the success of a software system. If there is low quality the system will never be accepted no matter how much functionality is offered. Low quality also means more maintenance effort and less reusability. Thus, it is in the interest of the user to define what he means under quality and to set his minimum quality fulfillment requirements before the project is started. To support this, *SoftCalc* has a separate quality characteristics table with the standard quality characteristics prescribed by the ISO–9126 standard plus user specific quality characteristics. In both cases the user is able to submit his expected quality fulfillment rate. The sum of the individual quality rates is then used to compute the quality multiplication factor.

## 4.1.6 Setting Project Influences

Once the product sizes and qualities have been set, either by importing them or setting them manually, the next step is to determine the project influences. Every cost estimation method has its own influence factors. The influence factors are different for COCOMO than for Function–Point or Object–Point. The Use Case Method has again another set of influence factors which only partly coincide with those of the other methods. As a result, *SoftCalc* must maintain a separate influence table for each method it supports.

In selecting the cost estimation effort he wants to use, the user is obliged to set the influences for that method. *SoftCalc* displays the appropriate table with preset default values and the user can overwrite them or leave them as they are. If he overwrites them he should consider what affect they will have on the estimation. Influence factors can alter the calculation significantly, depending on the method by +/– 50% .

## 4.1.7 Rating Project Resource Factors

Individual productivity can vary from the overall organizational productivity. Individuals have various degrees of availability and dependability and different productivity rates. Some persons are doubly more productive than others whereas some persons may only be available on a part-time basis. This has to be considered at the project level. The table of project resources allows the estimator to adjust the productivity of a project depending on the relative productivity of the resources assigned to it. There three types of resources

* Hardware resources
* Software resources and
* Human resources.

The user can judge the effect that each resource will have on the project productivity. Some resources will reduce it, others will increase it. It is not necessary to rate the allocated resources, but in some cases it may be helpful. If the estimator does not want to rate individuals, the table will not be used.

## 4.1.8 Analyzing Project Risks

The last step in preparing software estimation is to analyze the project risks. Every project involves risks, some more, some less, but the risks are there. It is up to the estimator to identify them and to assess what effect they may have on the project time and costs. To this end, *SoftCalc* provides a risk table which allows the user to identify the risks and to give in the risk probability as well as the risk exposure. From this the risk factor will be calculated. The sum of the individual risk factors is the overall project risk factor, which is then used to adjust the estimated effort.

## 4.1.9 Calculating Project Costs and Time

To calculate project costs and time *SoftCalc* offers no less than 8 different estimation methods:

* COCOMO – I
* COCOMO – II
* Data–Point
* Function–Point
* Object–Point
* Use Case–Point
* Test–Point
* Error Projection

Each method uses other size metrics and influence factors, but they all use the same quality factor, resource factor and risk factor. Every method will also have its own productivity data. First the size of the system is computed. Then the size is adjusted by the quality. From that an unadjusted effort is derived from the productivity table. The unadjusted effort is then adjusted by the influence factor, the resource factor and the risk factor to give a final effort in person months. From the final effort, the productivity and the degree of work distribution achievable, the time in calendar months is computed.

## 4.1.10 Projecting Error Rates

In the product table there is data on the overall system size, complexity and quality as well as on the extend of the test and the number of defects found. The size, complexity and quality metrics are furnished by *SoftAudit* by means of static analysis. The test metrics, including test coverage, are provided by the test tools. The number of defects is taken from the error reporting system. Using this data, *SoftCalc* can calculate the remaining error probability rate and give the number of error reports to be expected. Based on the past productivity in correcting errors, the total amount of effort required to budget for trouble shooting is projected.

## 4.1.11 Predicting Maintenance Costs

As an add on, *SoftCalc* uses the given requirement change rate and the projected error rate to predict the annual maintenance costs. For this purpose the system size is adjusted both by complexity and quality. The adjusted size is then multiplied by the change rate and the error rate to give an annual change size as prescribed by Boehm in the COCOMO method. By associating the change size with the maintenance productivity it is possible to estimate future maintenance costs, at least for the next release period, provided the maintenance productivity is accurate.

## 4.2 Inputs to SoftCalc

The inputs to *SoftCalc* come from two different sources

* User interface
* Import interface

The user interface is a Delphi GUI with multiple windows. The import interface allows multiple XML files to be read in.

## 4.2.1 SoftCalc User Interface

The *SoftCalc* user interface is a single, customizable graphical user interface with several alternate windows. There are two main windows for presenting and editing the contents of the data tables. The top window contains alternately the following product related tables:

* project / product table,
* object table,
* use case table,
* interface table,
* component table,
* test case table or
* system quality table.

The bottom window contains alternately the following project related tables:

* project influences table,
* project resource table,
* project risk table or
* project productivity table.

## 4.2.1.1 Project / Product Table

In the project/product table the user can submit the following header data:

* product name
* product identifier (this is the link to the import files)
* product type (standalone, integrated, distributed, internet,, embedded)
* project name
* project type (prototype, development, evolution, migration, integration)
* project technology (original, procedural, relational, object - oriented, web – based)
* maximum effort allowed for the project
* maximum time allowed for the project
* requirement change rate.

In addition he can give in or overwrite the following product metrics:

* Kilo lines of code
* Kilo Source statements
* Function–Points
* Data–Points
* Object–Points
* Use Case–Points
* Test–Points
* Expected errors
* Planned quality rating
* Planned complexity rating
* Planned annual change rate
* Planned test coverage
* No. Components
* No. Statements
* No. Procedures / Methods
* No. Procedures / Methods tested
* No. Files / Tables
* No. Data Attributes
* No. Data Attributes tested
* Actual Errors.

## 4.2.1.2 Object Table

In the object table the user can submit an object per line with the attributes:

* object name
* object type (data, view, table, internal file, external file)
* no. Relationships
* no. Keys
* no. Attributes
* change rate.

## 4.2.1.3 Use-Case Table

In the use–case table the user can submit a use–case per line with the attributes:

* use–case name
* use–case type (batch, online, cls, internet, service, embedded)
* no. objects
* no. steps
* no. inputs
* no. outputs
* change rate

## 4.2.1.4 Interface Table

In the interface table the user can submit an interface per line with the attributes:

* interface name
* interface type (file, panel, GUI, webpage, report, RPC, message)
* no. senders
* no. operations
* no. arguments
* no. results
* change rate

## 4.2.1.5 Test Case Table

In the test case table the user can submit a test case per line with the attributes:

* test case name
* test case type (requirements, design, code, interface, usage)
* no. inputs
* no. outputs
* no. objects
* no. steps
* change rate

## 4.2.1.6 Component Table

In the component table the user can submit a component per line with the attributes:

* component name
* component type (class, module, interface)
* no. lines of code
* no. statements
* no. functions or methods
* no. variables
* change rate

## 4.2.1.7 System Quality Table

In the system quality table the user can submit a quality goal per line with the attributes

* quality name
* quality type (usage, maintenance, reuse)
* quality weight
* median quality
* planned quality
* actual quality

## 4.2.1.8 Project Influence Table

In the project influence table the user can submit an influence factor per line with the attributes:

* influence factor name
* influence factor type (COCOMO–I, COCOMO–II, Data–Point, Function–Point, Object–Point, Test–Point, Usecase–Point)
* influence weight
* median influence
* planned influence
* actual influence

## 4.2.1.9 Project Resource Table

In the project resource table the user can submit a resource per line with the attributes:

* resource name
* resource type (human, hardware, software)
* dependability
* availability
* productivity
* cost

## 4.2.1.10 Project Risk Table

In the project risk table the user can submit a risk per line with the attributes:

* risk name
* risk type (organizational, technical, requirements, personnel)
* risk weight
* risk exposure
* risk probability
* risk reduction

## 4.2.1.11 Project Productivity Table

In the project productivity table the user submits a line per size measure of each previous project. The attributes here are:

* project name
* size measure (locs, statements, Function–Points, Data–Points, Object–Points, Use case–Points, Test–Points, Errors)
* project type (prototype, development, evolution, migration, integration)
* project technology (original, procedural, relational, object – oriented, web – based)
* no. size units
* no. person months required to complete project
* no. calendar months required to complete project
* total costs of project in kilo currency units

## 4.2.2 SoftCalc Import Files

SoftCalc can import the following 9 types of XML files

* productivity file
* requirement entity file
* design entity file
* code entity file
* data entity file
* interface entity file
* test case file
* test data file
* test protocol file

## 4.2.2.1 Productivity File (ProdTab)

The productivity file can be generated from an Excel table or exported from the productivity table in another *SoftCalc* database. It contains the same 8 attributes that are contained in the internal productivity table. There is a data group for each project size measure with a data element for each attribute. The elements correspond to the lines of the table. Thus, there exists a 1:1 relationship between the Excel table where the data originates, the XML file by which the data is transported and the *SoftCalc* relational table where the data is stored.

## 4.2.2.2 Requirement Entity File (REntity)

The requirement entity file is produced from the requirements text analysis by the Text Analyzer. (See Text Analyzer documentation). It contains four entity types

* use cases
* objects
* interfaces and
* test cases

Each entity type is a data group in the XML file containing an element for each attribute. The attributes correspond 1:1 to the attributes in the *SoftCalc* tables.

## 4.2.2.3 Design Entity File (UEntity)

The design entity file is produced from the UML Audit tool. (See Chapter 1.) It contains the same four entity types as the requirement entity file plus a

* component entity

The use case entities are generated from the use case diagrams. The object entities are generated from the class diagrams. The interface entities are generated from the class interaction diagrams. The test case entities are generated from the sequence, activity and state diagrams. The component entities are created from the packaging diagrams. Each entity type is a data group in the XML file containing an element for each attribute. The attributes correspond 1:1 to the attributes in the *SoftCalc* tables.

## 4.2.2.4 Code Entity File (CEntity)

The code entity file is produced by the *SoftAudit* tools which process programming languages. (See Chapter 1). It contains three types of code entities

* modules
* classes and
* interfaces

Modules are extracted from the procedural source code. Classes are extracted from the object–oriented source code. Interfaces are extracted from both source types where files, maps, reports and messages are defined. Each code entity is a data group corresponding to an entry in the component table. Each class entity corresponds to an entry in the object table. Each interface entity corresponds to an entry in the interface table.

The code entity file also has a summary group at the end which contains the total number of lines, statements, procedures, Functions–Points, Data–Points, and Object–Points for the system as a whole plus the quality and complexity rates of that system. These metrics are loaded into the product data table.

## 4.2.2.5 Data Entity File (DEntity)

The data entity file is produced by the *SoftAudit* tools which process database description languages. (See Chapter 1). It contains a single entity type

* data object

created from the database schema analysis. The attributes of this entity correspond 1:1 to the attributes of the *SoftCalc* object table.

## 4.2.2.6 Interface Entity File (IEntity)

The interface entity file is produced by the *SoftAudit* tools which process interface description languages. (See Chapter 1). It contains a single entity type

* data interface

created from the interface description analysis. The attributes of this entity correspond 1:1 to the attributes of the *SoftCalc* interface table.

## 4.2.2.7 Test Case File (TEntity)

The test case file is generated by the Test Case Analyzer tool. (See Test Analyzer Documentation). It contains a single entity type

* test case entity

The test case entity is a data group containing the four metrics of a test case – number of inputs, number of outputs, number of objects accessed by the test case and number of steps executed by the test case. These metrics correspond 1:1 to the attributes of the entities in the *SoftCalc* test case table.

## 4.2.2.8 Test Data File (XEntity)

The test data file is generated by the Data Test tool. (See Test Tools Documentation). It contains metrics for the product data table. These are:

* number of records or table rows processed,
* number of incorrect records or rows,
* number of data fields defined in the record or table,
* number of data fields tested,
* number of incorrect data fields,
* ratio of fields tested (data coverage).

These data items flow into the product table where they are used to project the number of remaining errors and to assess the dynamic quality of the software system.

## 4.2.2.9 Test Protocol File (PEntity)

The test protocol file is created by the Test Documentor tool. (See Test Tools Documentation). It contains additional metrics for the product data table. These are:

* number of components tested,
* number of lines in the tested components,
* number of statements in the tested components,
* number of methods and procedures in the tested components,
* number of method and procedures actually tested,
* number of errors discovered,
* test coverage ratio.

These test metrics flow into the product table where they are used to project the number of remaining errors and to assess the dynamic quality of the software system.

## 4.3 Outputs from SoftCalc

There are three types of output produced by the *SoftCalc* tool:

* displays in the user interface,
* reports in the print spool, and
* exports from the database.

## 4.3.1 Displays

The following outputs are displayed on the *SoftCalc* user interface

* size measures
* size calculations
* effort calculations
* calculation results.

The size measures are the number of the key entity types – use cases, objects, interfaces, components and test cases.

The size calculations are those intermediate metrics computed from the key entities which indicate the size of the planned or actual product. These are:

* unadjusted size in Klocs, KDSIs, Function-Points, Data-Points, Usecase-Points, Test-Points or defect counts.
* complexity factor
* complexity adjusted size
* quality factor
* quality adjusted size
* final adjusted size in Klocs, KDSIs, Function–Points, Data–Points, Object–Points, Usecase–Points, Test–Points or defect counts.

The effort calculations are the different intermediate results computed from the productivity table and adjusted by the project influences, resources and risks.

* unadjusted effort as taken from the productivity table
* influence factor
* influence adjusted effort
* resource factor
* resource adjusted effort
* risk factor
* risk adjusted effort
* final effort in person months.

The calculation results are

* minimum effort required to complete the project
* minimum time required to complete the project
* minimum costs required to pay for the project
* optimal staff size for the project
* minimum annual effort required to maintain the product.

## 4.3.2 Printed Reports

The printed reports are lists of those tables used to make the project calculations. The user is given a menu of all tables:

* product data table
* object table
* use–case table
* interface table
* component table
* test case table
* quality table
* influence table
* resource table
* risk table
* productivity table.

The user can select which tables he wants to print out. These tables are then listed out with a line per entry and a column per attribute. Furthermore, the user can also print out the results of a calculation. By selecting the print function calculate in conjunction with an estimation method a one page report on the calculation results is generated with the

* size measures
* size calculations
* effort calculations and
* calculation results

of that particular estimation method. This print function can then be repeated for every estimation method the user wants to use.

## 4.3.3 Exports

There are two XML export file types provided by the *SoftCalc* tool:

* Productivity Export File
* Project Export File

The productivity export file contains only the contents of the productivity table. There is an XML data group for each reference project with a data element for each productivity measure of that project. This file can be used to transport the productivity data from on *SoftCalc* database to the other.

The project export file contains the contents of all tables in the project database. There is an Xml data group for each table with a subgroup for each entry in the table and a data element for each attribute of that entry. This file is used to save the contents of the *SoftCalc* database and to transport project data from on database to another. It can, of course, also be used to transport project data from *SoftCalc* to other tools.

## 4.4 Using SoftCalc

Estimating the costs of a software project is not a task which can be performed by an amateur. *SoftCalc* helps to simplify the task in three ways:

* By providing a frame of reference,
* By guiding the user to a result and
* By making data available.

The user must still be able to recognize infeasible results and to explain how they came to be. That is one reason for having so many different calculation methods. When one method differs significantly from the other methods, this is a sign that some data is inconsistent.

The user should search out and correct the data, then repeat the estimation until the calculation results converge. The user should also know what estimation methods fit best to the type of project he is trying to estimate. It is not appropriate to estimate an object–oriented evolution project with Function–Points, just as it is not appropriate to estimate a prototype project with Object–Point or a migration project with Use–case points. The size metrics used must be applicable to the type of product being sized.

Finally, the user must feed the tool with accurate, reliable data. No tool, no matter how sophisticated, can produce accurate results from inaccurate inputs. Garbage in, garbage out. The input data should be screened and verified before it is used. The greatest danger in using *SoftCalc* is that the user will feed it with incomplete and inaccurate data. In so far as the data is formally correct, *SoftCalc* will produce some result, and if the data is consistent, the results of the various methods will be similar. The tool can not recognize that requirements or components are missing. Therefore, care must be taken to ensure that the data tables are as complete and accurate as possible.

Literature: See Software in Zahlen von Sneed, Baumgartner & Seidl, Hanser Verlag, 2010