LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

Master's Thesis

THE DEVELOPMENT OF AN INTERNAL TESTING PROCESS FOR A BLUETOOTH PRODUCT

The council of the Department of Information Technology approved the subject of the thesis on October 10, 2001.

Supervisor: Group Manager Timo Kyntäjä
Inspector: Professor Pekka Toivanen

Espoo, December 20, 2001

Katja Pulkkinen
Eerikinkatu 29 B 28
FIN-00180 Helsinki
Finland

ABSTRACT

Author:	Katja Pulkkinen
Subject:	**The Development of an Internal Testing Process for a Bluetooth Product**
Department:	Information technology
Year:	2002
Place:	Espoo
Master's Thesis:	Lappeenranta University of Technology. 70 pages, 19 figures, 15 tables and 3 appendices.
Supervisor:	Professor Pekka Toivanen
Keywords:	testing, testing process, usage-based testing, Bluetooth qualification

BlueGiga Technologies is a small start-up company adapting Bluetooth technology. A testing process was needed to complement their product development process. The creation of the testing process was a challenge, because of the new technology, company's young age, and the integration of hardware and software components in the products.

The testing started with the evaluation of a standard method to document the tests. After this, BlueGiga's software development process was studied and positioned in the field of existing software development processes. At the same time, the requirements for testing imposed by the Bluetooth technology and qualification process were studied. As a result of this, TTCN was tried in defining a human readable test case. The suitability of usage-based testing for the testing of Wireless Remote Access Platform (WRAP) product family's different usage scenarios was evaluated. This was done by applying it to the testing of Man-to-Machine usage scenario.

Based on the information and experience acquired during the tasks described above, a testing process was created. The testing process covers the unit, integration and system testing with emphasis on system testing. The process also defines the person or persons responsible for different levels of testing.

TIIVISTELMÄ

Tekijä: Katja Pulkkinen

Nimi: **Sisäisen testausprosessin kehittäminen Bluetooth -tuotteelle**

Osasto: Tietotekniikan osasto

Vuosi: 2002

Paikka: Espoo

Diplomityö: Lappeenrannan teknillinen korkeakoulu. 70 sivua, 19 kuvaa, 15 taulukkoa ja 3 liitettä.

Tarkastaja: Professori Pekka Toivanen

Hakusanat: testaus, testausprosessi, käyttötapauksiin perustuva testaus, Bluetooth -kvalifikaatio

BlueGiga Technologies on uusi Bluetooth -teknologiaa soveltava pk-yritys. Yrityksen tuotekehitysprosessia täydentämään tarvittiin testausprosessi. Testausprosessin luominen oli haastavaa, koska Bluetooth -teknologia on uutta ja yritys on vielä nuori. Lisäksi se integroi kovo- ja ohjelmistokomponentteja tuotteissaan.

Testaus aloitettiin evaluoimalla standardinmukaista tapaa dokumentoida testit. Tämän jälkeen tutkittiin BlueGigan ohjelmistokehitysprosessin suhdetta olemassa oleviin ohjelmistokehitysprosesseihin. Samanaikaisesti perehdyttiin Bluetooth -kvalifikaation testaukselle asettamiin vaatimuksiin. Tämän seurauksena TTCN:ää kokeiltiin helppolukuisen testitapauksen määrittelyssä. Käyttötapauksiin perustuvan testauksen sopivuutta Wireless Remote Access Platform:in (WRAP) testaamiseen arvioitiin kokeilemalla sitä Man-to-Machine -käyttötapauksen testaamisessa.

Yllämainittujen tehtävien aikana kerätyn tiedon ja hankittujen kokemusten pohjalta laadittiin testausprosessi, joka kattaa yksikkö-, integraatio- ja järjestelmätason testauksen. Painopiste on järjestelmätason testauksessa. Prosessi määrittelee myös vastuuhenkilön tai
-henkilöt eri testaustasoille.

PREFACE

I would like to thank the following individuals for making this thesis possible: my supervisor Timo Kyntäjä for his continuing interest in my thesis work which was invaluable for the completion of it. Tom Nordman from BlueGiga Technologies for his support and interest in my work. All my colleagues and workmates both at VTT and BlueGiga Technologies for creating an encouraging and inspiring working atmosphere. My inspector Pekka Toivanen for excellent writing tips.

Many thanks also to the students, lecturers, assistants and other personnel at Lappeenranta University of Technology for memorable years of work, studies and student life. And last, but not least, I thank my parents and my boyfriend Kimmo for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DISCLAIMER

ABBREVIATIONS

API	Application Programming Interface: a set of subprograms that application programs may use to request and carry out lower-level services.
ASN.1	Abstract Service Notation One. a standard, flexible method that (a) describes data structures for representing, encoding, transmitting, and decoding data, (b) provides a set of formal rules for describing the structure of objects independent of machine-specific encoding techniques.
ASP	Abstract Service Primitive: a message exchanged between protocol layers through a service access point.
BER	Bit Error Rate: used to measure the quality of a link.
BLUCE	BlueGiga Core Engine: provides the base over which customer applications are built.
BQB	Bluetooth Qualification Body: an individual person recognized by the BQRB to be responsible for checking declarations and documents against requirements, verifying the authenticity of product test reports, and listing products on the official database of Bluetooth qualified products.
BQRB	Bluetooth Qualification Review Board: is responsible for managing, reviewing and improving the Bluetooth Qualification program. The Bluetooth SIG promoter companies appoint delegates to the BQRB.
BQTF	Bluetooth Qualification Test Facility: a test facility officially authorized by the BQRB to test Bluetooth products.
EUT	Equipment Under Test.
GAP	Generic Access Profile: defines a basic set of procedures that all Bluetooth devices use both in handling connections (e.g., timers) and at the user interface level (e.g., naming conventions).
HCI	Host Controller Interface: the host which links a Bluetooth host to a Bluetooth module. Data, commands, and events pass across this interface.
HTTP	Hyper Text Transfer Protocol: an application protocol providing means to transfer hypertext documents between servers and clients.
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IP	Internet Protocol: a Protocol that provides addressing, routing, segmentation, and re-assembly.
IrDA	Infrared Data Association: an organization that defines the infrared communications protocol used by many laptops and mobile cellular phones to exchange data at short ranges.
IUT	Implementation Under Test.
LAN	Local Area Network: a computer network located on a user's premises within a limited geographical area.
LAP	LAN Access Point: one of the Bluetooth profiles used to define how products can be made which allow devices to use Bluetooth links to access a LAN.
L2CA	Logical Link Control and Adaptation: the layer of the Bluetooth stack which implements L2CAP. This provides segmentation and reassembly services to allow large packets to pass across Bluetooth links; also provides multiplexing for higher layer protocols and services.

L2CAP	Logical Link Control and Adaptation Protocol, see L2CA above.
LCD	Liquid-Crystal Display: A display device that uses the change of the optical properties of a liquid crystal when an electric field is applied.
LT	Lower Tester.
MSC	Message Sequence Chart.
OSI	Open Systems Interconnection: an abstract description of the digital communications between application processes running in distinct systems. The model employs a hierarchical structure of seven layers. Each layer performs value-added service at the request of the adjacent higher layer and, in turn, requests more basic services from the adjacent lower layer.
PAN	Personal Area Network: the user is surrounded or wearing electronic devices which either require data input or provide data output.
PDA	Personal Digital Assistant: a small handheld computing device such as a Palm Pilot.
PCO	Point of Control and Observation: the point where RECEIVE and SEND events for Abstract Service Primitives (ASPs) and Protocol Data Units (PDUs) can be observed.
PDU	Protocol Data Unit: in Open Systems Interconnection (OSI), a block of data specified in a protocol of a given layer and consisting of protocol control information of that layer, and possibly user data of that layer i.e. messages changed horizontally between similar layers of a protocol stack.
PPP	Point to Point Protocol: an Internet protocol which provides a standard way of transporting datagrams from many other protocols over point to point links.
PRD	Qualification Program Reference Document: the reference to specify the functions, organization and processes inside the Bluetooth Qualification program.
RF	Radio Frequency.
RFCOMM	Protocol for RS-232 serial cable emulation.
SAP	Service Access Point: in open systems Interconnection (OSI) layer, a point at which a designated service may be obtained.
SDP	Service Discovery Protocol: a Bluetooth protocol which allows a client to discover the devices offered by a server.
SIG	Special Interest Group, the Bluetooth Special Interest Group: a group of companies driving the development of Bluetooth technology and bringing it to market.
SPP	Serial Port Profile: a Bluetooth specification for how serial ports should be emulated in Bluetooth products.
SUT	System Under Test.
TCP	Transport Control Protocol: part of the Internet protocol suite which provides reliable data transfer.
TCP/IP	Transport Control Protocol/Internet Protocol: part of the Internet protocol suite which provides reliable data transfer.
TTCN	Tree and Tabular Combined Notation: standardized language for describing 'black-box' tests for reactive systems such as communication protocols and services.

UDP	User Datagram Protocol: a packet-based, unreliable, connectionless transport protocol which works over Internet Protocol (IP).
URL	Uniform Resource Locator: defined by RFC1738, this is a standard way of writing addresses on the Internet; for example, http://www.bluetooth.com is a URL.
UI	User Interface: The part of a system with which a user interacts.
UT	Upper Tester.
UUID	Universally Unique IDentifier: a 128-bit number derived by a method which guarantees it will be unique.
WRAP	Wireless Remote Access Platform: BlueGiga Technologies' platform for developing new wireless applications or adding wireless connectivity to existing devices.
XP	Extreme Programming: a software development process model.

1 INTRODUCTION

The development of a testing process for a small company using new technology is a challenging task. This is due to the intensive product development cycles and a lack of existing testing practices, both at the company level and in the industry in general. A testing process was needed at BlueGiga Technologies which is a small start-up company adapting Bluetooth technology.

When using a new technology, also the testing standards for that technology evolve simultaneously with the product development process. At the beginning of this project, there was only one thesis work [Hel01, Toi01] done related to the testing of Bluetooth technology. Therefore, there was a need to clarify the issues related to Bluetooth testing and qualification.

The aim of this work is to develop a process for testing that could be used at BlueGiga Technologies. The process has to take into account the requirements for testing and qualifying Bluetooth technology. The main emphasis is on the process, even if some techniques used at different phases of testing are also discussed.

Testing is part of the larger software development process. Those processes that can be identified in the BlueGiga's software development are explained. After this, the emphasis is on testing part of the process. The different levels of testing and who should be doing them are explored. Some existing testing standards and practices, e.g. IEEE standards, TTCN and usage-based testing, are studied.

Those parts of Bluetooth technology that are necessary in understanding the BlueGiga Technologies' Wireless Remote Access Platform (WRAP) at a general level are described. BlueGiga Technologies is briefly introduced. Experiences on testing are reported and the testing process is outlined based on these experiences.

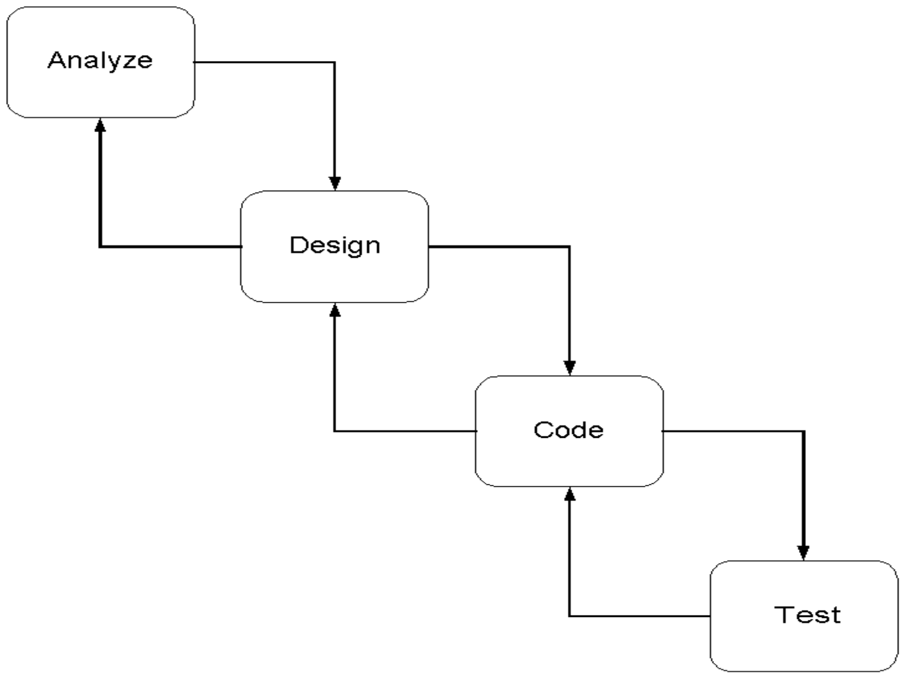
2 SOFTWARE DEVELOPMENT PROCESS MODELS

This chapter is a brief summary of some basic software development models. There exists a lot more of them [Bha00, Kan97], but only those that are of relevance to the BlueGiga Technologies' software development process at the moment are discussed here. Also, a more detailed description of each development process model could be provided [Kan97, Pre00, Wel01], but only those parts that are essential to the BlueGiga Technologies' process are examined here.

2.1 The Waterfall Development Model

The waterfall development model, sometimes also called the linear sequential model or the lifecycle model, is the classic software process model developed in the 1970s [Kan97, p. 14; Pre00, p. 26]. It's most important lesson to the modern programmers is that the first thing to do in software development is to specify what the software is supposed to do [Kan97, p. 14]. The process consists of the following activities shown in Figure 1: analysis, design, coding and testing. [Kan97, p. 14; Pre00, p. 28]

Figure 1. Waterfall Model.



From the waterfall development model we learned that proper gathering and defining of system requirements is important [Kan97, p. 14]. It is also the base of testing, because testing measures actual behavior against required behavior [IEEE86, p. 19] Poorly defined requirements lead to problems in testing. Carefully defined requirements save a lot of work at the later phases of software development, including testing.

2.2 The Prototyping Approach

The waterfall "model assumes that requirements are known, and that once requirements are defined, they will not change [Kan97, p. 19]". This is not usually the case when developing products for customers. Sometimes the requirements are not even known in the beginning. The changing of requirements during the software development process is more a rule than an exception.

Various software development process models have been developed that attempt to deal with customer feedback on the product to assure that it satisfies the requirements. Each of these models provides some form of prototyping. [Kan97, p. 20]

There are two basic types of prototyping: throwaway prototyping and evolutionary prototyping [Kan97, p. 21-22]. In BlueGiga Technologies' case the prototyping is evolutionary. "A prototype is built based on some known requirements and understanding. The prototype is then refined and evolved instead of thrown away." [Kan97, p. 22] It is not reasonable nor economical to throw away prototypes for complex applications. Evolutionary prototypes are based on prioritized requirements. This is reasonable for complex applications where there are lots of requirements. [Kan97, p. 22]

Prototyping is not a complete software development process in itself. It is usually used together with some other software development process. [Kan97, p. 50] In BlueGiga's case the other software development process is the spiral model.

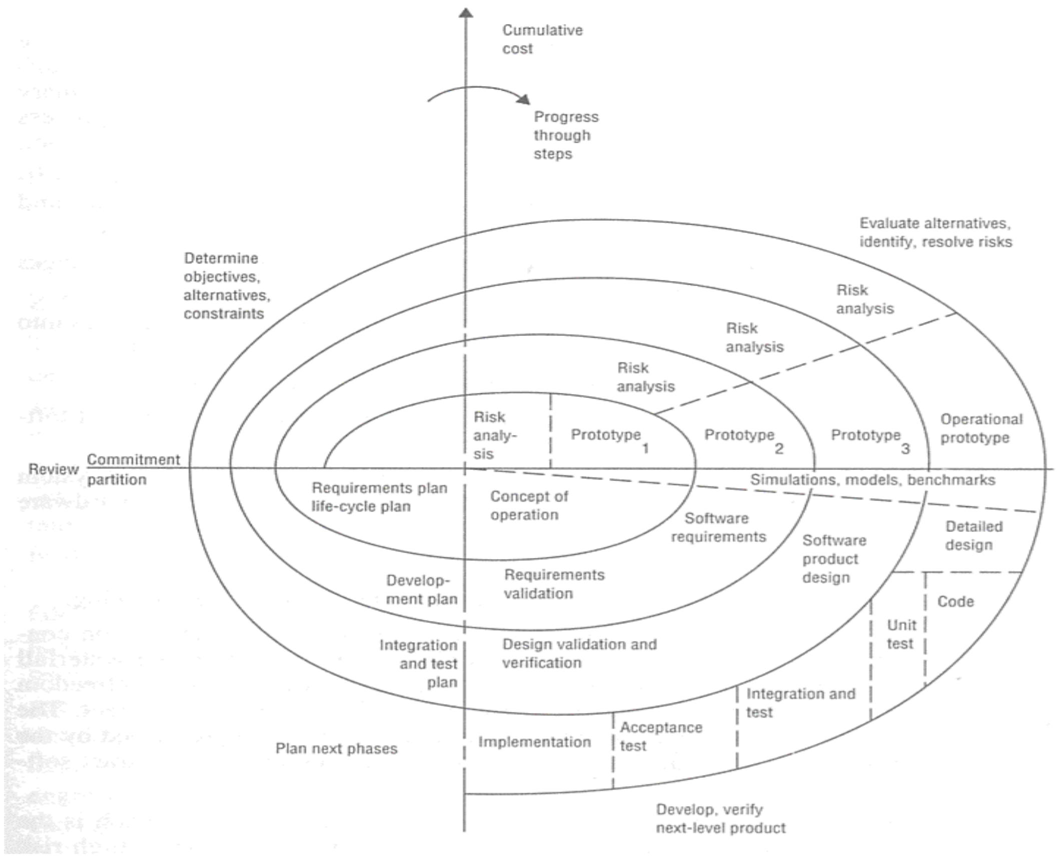
Evolutionary prototyping uses prioritized requirements. Therefore, detailed requirements planning is the key in developing a prototype implementing the most important parts of the system. Being able to react to changes in requirements is essential when developing products to customers. This is where prototyping is extremely useful in BlueGiga's case. Prototypes can also be used as demos for potential customers.

2.3 The Spiral Model

The spiral model relies heavily on prototyping and risk management. It is much more flexible than the waterfall model. [Kan97, p. 22] "The underlying concept of the model is that for each portion of the product and for each of its levels of elaboration, the same sequence of steps (cycle) is involved [Kan97, p. 22]". The spiral model is iterative because it takes advantage of prototyping. It is also incremental because there are several cycles which all contain the same steps. These are the respects in which it is better than the waterfall model. [Kan97 , p. 22, Pre00, p. 35]

"The radial dimension in Figure 2 represents the cumulative cost incurred to date in accomplishing the steps. The angular dimension represents the progress made in completing each cycle of the spiral. As indicated by the quadrants in the figure, the first step of each cycle of the spiral is to identify the objects of the portion of the product being elaborated, the alternative means of implementation of this portion and the constraints imposed on the application of alternatives. The next step is to evaluate the alternatives relative to the object and constraints, and to identify the associated risks and resolve them. Risk analysis and the risk-driven approach, therefore, are key characteristics of the spiral model, versus the document-driven approach of the waterfall model." [Kan97, p. 22]

Figure 2. Spiral Model of the Software Process. (Copyright© 1988 IEEE)



(Source: B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, May 1988, pp. 61-72. Permission to reprint obtained from IEEE.)

The spiral model accommodates preparation for life-cycle evolution, growth, and changes of the software product [Kan97, p. 24]. "It provides a viable framework for integrating hardware-software system development" [Kan97, p. 24] which is important for BlueGiga. "The risk-driven approach can be applied to both hardware and software [Kan97, p. 24]."

The spiral model has also its difficulties. One of these "is how to achieve the flexibility and freedom prescribed by the spiral model without loosing accountability and control for contract software." [Kan97, p. 24]

The model relies on risk management expertise. [Kan97, p. 24] "The risk-driven approach is the backbone of the model. The risk-driven specification carries high-risk elements down to a great level of detail and leaves low-risk elements to be elaborated in later stages. However, an inexperienced team may also produce a specification just the opposite: a great deal of detail for the well understood, low-risk elements and little elaboration of the poorly understood high risk elements. Another concern is that a risk-driven specification is also people dependent. In a case where a design produced by an expert is to be implemented by non-experts, the expert must furnish additional documentation." [Kan97, p. 24-25]

“The spiral model describes a flexible and dynamic process model that can be utilized to its fullest advantage by experienced developers. However, for non-experts and especially for large-scale projects, the steps in the spiral must be further elaborated and more specifically defined so that consistency, tracking, and control can be achieved. Such elaboration and control are especially important in the area of risk analysis and risk management.” [Kan97, p. 25]

The spiral model provides a viable framework for integrating hardware-software system development [Kan97, p. 24]. “Its range of options accommodates the good features of existing software process models, whereas its risk-driven approach avoids many of their difficulties [Kan97, p. 24].” Risk management is essential when developing an innovative product based on a technology that is still in the maturing stage.

2.4 Extreme Programming

“Now in the 21st century, many of the ‘old’ heavy-weight software development processes’ rules are hard to follow, procedures are complex and not well understood and the amount of documentation written in some abstract notation is way out of control. XP is one of the new light-weight software development processes. Its’ goal is to improve software projects through communication, simplicity, feedback and courage.” [Wel01]

“XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. With this foundation XP programmers are able to courageously respond to changing requirements and technology.” [Wel01]

“XP was created in response to problem domains whose requirements change. In many software environments dynamically changing requirements is the only constant. This is when XP is claimed to succeed while other methodologies do not.” [Wel01]

“XP was also set up to address the problems of project risk. If the customers need a new system by a specific date the risk is high. If that system is a new challenge for the software group the risk is even greater. If that system is a new challenge to the entire software industry the risk is greater even still. The XP practices are set up to mitigate the risk and increase the likelihood of success.” [Wel01] Products based on Bluetooth technology might be considered a new challenge to the entire industry.

“XP is set up for small groups of programmers. Between 2 and 10. The programmers can be ordinary, they do not need to have a Ph.D. to use XP. But XP cannot be used on a project with a huge staff. It should be noted that on projects with dynamic requirements or high risk a small team of XP programmers might be more effective than a large team anyway.” [Wel01] At BlueGiga Technologies there are currently four programmers and none of them is a Ph.D.

“XP requires an extended development team. The XP team includes not only the developers, but the managers and customers as well, all working together elbow to elbow.” [Wel01] The product development at BlueGiga is going to be continued with customer pilot projects where customers have the key role in defining functional requirements.

“Another requirement is testability. You must be able to create automated unit and functional tests [Wel01].” This is where the improvement could be made at BlueGiga. According to XP, “tests are created before the code is written, while the code is written, and after the code is written” [Wel01]. The tests that should be created before the code is written are the unit tests. [Wel01] While the code is written the integration tests can be created. After the code is written, tests that reuse existing code from the prototype can be created.

The XP is suitable for projects where requirements change, risks are high due to new technology, and programming staff is relatively small [Wel01]. This is the situation at BlueGiga at the moment. XPs’ requirement of an extended development team is not a problem, because the management and programmers already work together “elbow to elbow” as Wells suggests. If making the customers part of the development process succeeds is to be seen in the future pilot projects. Another important requirement is testability and that is what we are going to implement in this thesis.

3 SOFTWARE TESTING

“Experienced software developers often say, ‘Testing never ends, it just gets transferred from you to your customer. Every time your customer uses the program a test is being conducted.’ By applying test case design, the software engineer can achieve more complete testing and thereby uncover and correct the highest number of errors before the ‘customer’s tests’ begin.” [Pre00, p. 460]

3.1 Test Design

“Bug prevention is testing’s first goal [Bei90, p. 3].” A prevented bug is better than a detected bug [Bei90, p. 3]. “More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done in order to create a useful test can discover and eliminate bugs before they are coded – indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.” [Bei90, p. 3] Gelperin and Hetzel advocate ‘Test then code’ [Bei90].” This kind of thinking is typical also in extreme programming. “The ideal test activity would be so successful at bug prevention that actual testing would be unnecessary because all bugs would have been found and fixed during test design [Bei90, p. 3-4].”

“Unfortunately, we cannot achieve this ideal. Despite our effort, there will be bugs because we are human. To the extent that testing fails to reach its primary goal, bug prevention, it must reach its secondary goal, bug discovery. Bugs are not always obvious. A bug is manifested in deviation from expected behavior. A test design must document expectations, the test procedure in detail, and the results of the actual test – all of which are subject to error. But knowing that a program is incorrect does not imply knowing the bug. Different bugs can have the same manifestations, and one bug can have many symptoms. The symptoms and the causes can be disentangled only by using many small detailed tests.” [Bei90, p. 4]

“The test design phase of programming should be explicitly identified. Instead of ‘design, code, desk check, test, and debug’ the programming process should be described as: ‘design, test design, code, test code, program desk check, test desk check, test debugging, test execution, program debugging, correction coding, regression testing, and documentation’.” [Bei90, p. 7] Under this classification scheme, the component parts of programming are comparable both in size and costs. Now it is obvious that at least 50 % of a software project’s time is spent on testing. Realizing this makes it more likely that testing will be done, even if budget is small and schedule is tight. [Bei84, p. 4; Bei90, p. 7]

“The primary objective for test case design is to derive a set of tests that have high likelihood of uncovering errors in the software. To accomplish this objective, two different categories of test case design techniques are used: black-box testing and white-box testing.” [Pre00, p. 460]

In black-box testing, the tests are designed from a functional point of view. The system is treated as a black-box which is subjected to inputs, and its outputs are verified for conformance to specified behavior [Bei90, p.

10]. “The software’s user should be concerned only with functionality and features, and the program’s implementation details should not matter [Bei90, p. 10].”

“Black-box testing techniques focus on the information domain of the software, deriving test cases by partitioning the input and output domain of a program in a manner that provides thorough test coverage. Equivalence partitioning divides the input domain into classes of data that are likely to exercise specific software function. Boundary value analysis probes the program’s ability to handle data at the limits of acceptability. Orthogonal array testing provides an efficient systematic method for testing systems with small numbers of input parameters.” [Pre00, p. 460]

Unlike black-box testing, white-box testing looks at the implementation details. Such things as programming style, control method, source language, database design and coding details are central in structural white-box testing [Bei90]. “White-box tests exercise the program’s control structure [Pre00, p. 460]. “Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been exercised [Pre00, p. 460].” Basis path testing makes use of program graphs, or graph matrices, to derive a set of linearly independent tests that will ensure coverage [Pre00, p. 460]. “Condition and data flow testing further exercise program logic, and loop testing complements other white-box testing techniques by providing a procedure for exercising loops of varying degrees of complexity.” [Pre00, p. 460]

“Hetzel describes white-box testing as ‘testing in the small’. His implication is that the white-box tests are typically applied to small program components (e.g. modules, or small groups of modules), black-box testing, on the other hand broadens the focus and might be called ‘testing in the large’.” [Pre00, p. 460]

“There’s no controversy between the use of structural versus functional tests: both are useful, both have limitations, both target different kinds of bugs. Functional tests can, in principle, detect all bugs but would take infinite time to do so. Structural tests are inherently finite but cannot detect all errors, even if completely executed. The art of testing, in part, is in how you choose between structural and functional tests.” [Bei90, p. 11]

3.2 Levels of Testing

“We do three distinct kinds of testing on a typical software system: unit/component testing, integration testing, and system testing [Bei90, p. 20].” The objectives of each class of testing are different and, therefore, also the mix of test methods used differs [Bei90, p. 20-21]. Both structural and functional test techniques can and should be applied in all three phases; however there is a higher concentration of structural techniques for units and components and conversely, system testing is mostly functional [Bei90, p. 428].” In order to achieve adequate testing this question needs to be answered: ‘Who should be responsible for what kind of tests?’ [Bei90, p. 428].

3.2.1 Unit and Component Testing

A unit is the smallest testable piece of software [Bei90, p. 21]. “A unit is usually the work of one programmer and it consists of several hundred or fewer lines of source code. Unit testing is the testing we do to show that the unit does not satisfy its functional specification and/or that its implemented structure does not match the intended design structure. When our tests reveal such faults, we say that there is a unit bug.” [Bei90, p. 21]

“A component is an integrated aggregate of one or more units. A unit is a component, a component with subroutines it calls is a component, etc.” [Bei90, p. 21] By this recursive definition, a component can be anything from a unit to an entire system [Bei90, p. 21]. “Component testing is the testing we do to show that the component does not satisfy its functional specification and/or that its implemented structure does not match the intended design structure. When our tests reveal such faults, we say that there is a component bug.” [Bei90, p. 21]

“Unit and component testing is firmly in the programmer’s hands. In an attempt to improve the software development process, there have been experiments aimed at determining whether testing should be done by programmer’s and, if so, to what extent. These experiments have ranged from no programmer component testing at all to total control of all testing by the designer. The consensus, as measured by successful projects (rather than by toy projects in benign environments) puts component testing and responsibility for software quality firmly in the programmer’s hands.” [Bei90, p. 429]

3.2.2 Integration Testing

“Integration is a process by which components are aggregated to create larger components. Integration testing is testing done to show that even though the components were individually satisfactory, as demonstrated by the successful passage of component tests, the combination of components are incorrect or inconsistent. For example, components A and B have both passed their component tests. Integration testing is aimed at showing inconsistencies between A and B. Examples of such inconsistencies are improper call or return sequences, inconsistent data validation criteria, and inconsistent handling of data objects. Integration testing should not be confused with testing integrated objects, which is just higher level of component testing. Integration testing is specifically aimed at exposing the problems that arise from the combination of components. The sequence, then, consists of component testing for components A and B, integration testing for the combination of A and B, and finally, component testing for the ‘new’ component (A,B).” [Bei90, p. 21]
“Integration is a no-man’s land. Part of the problem is that in many groups it is hopelessly confused with system testing – where ‘integration testing’ is used synonymously with ‘testing the integrated system’. With such semantic confusion, there can be no integration testing worthy of the name.” [Bei90, p. 430]
Beizer suggests, that programmer’s do the unit and component level tests to the units and components they have coded. The basic set of units and components, e.g. the back-bone of the system, is then integrated by one of the programmers. Formal integration test design and testing is performed by a programmer, who is responsible for quality assurance at this stage. Then the integrated component is returned to one of the

programmers, who participated in its' coding, for component-level functional and structural testing. [Bei84, p. 162-163]

3.2.3 System Testing

"A system is a big component. System testing is aimed at revealing bugs that cannot be attributed to components as such, to the inconsistencies between components, or to the planned interactions of components and other objects. System testing concerns issues and behaviors that can only be exposed by testing the entire integrated system or a major part of it." [Bei90, p. 22]

"A fully integrated system that has been thoroughly tested at every level is not enough [Bei84, p. 165]." "If all elements from unit to system have been thoroughly tested, functionally and structurally, and no known incompabilities between elements remain, what then is there left to test at the system level that has not already been tested [Bei84, p. 165]?" Approximately half of the testing and quality assurance (QA) effort remains to be expended in the following tests:

1. System-level functional verification by the programming staff and/or quality assurance.
2. Formal acceptance test plan design and execution thereof by the buyer or a designated surrogate.
3. Stress testing – attempting to 'break' the system by stressing all of its resources.
4. Load and performance testing – testing to conform that performance objectives are satisfied.
5. Background testing – re-testing under a real load instead of no load – a test of proper multi-programming multi-tasking systems.
6. Configuration testing – testing to assure that all functions work under all logical/physical device assignment combinations.
7. Recovery testing – testing that the system can recover from hardware and/or software malfunctions without loosing data or control.
8. Security testing – testing to confirm that the system's security mechanism is not likely to be breached by illicit users. [Bei84, p. 165]

"System-level functional testing, formal acceptance testing, and stress testing are required for every system. Background testing is required for all systems but the simplest uniprocessing batch systems. Background testing is usually not required (but is desirable) for application programs run under an operating system. Load and performance testing is required for all on-line systems. Configuration testing is required for all systems in which the program can modify the relation between physical and logical devices and all systems in which backup devices and computers are used in the interest of system reliability. Recovery testing is required for all systems in which data integrity despite hardware and/or software failures must be maintained, and for all systems that use backup hardware with automatic switchover to the backup devices. Security testing is required for all systems with external access." [Bei84, p. 166]

"System testing is less firmly in the independent tester's hands. There are three stages of organizational development with respect to system testing and who does it." [Bei90, p. 429]

"Stage 1: Preconscious – The preconscious phase is marked by total ignorance of independent testing. It's business as usual, as it's been done for decades. It works for small projects (four to five programmers) and rapidly becomes catastrophic as project size ad program complexity increases. Today, most software development is still dominated by this state of happy ignorance." [Bei90, p. 429]

"Stage 2: Independent testing – Independent testing marks most successful projects of more than 50K lines of new code. Independent system testing is a feature of many government software development specifications. This testing typically includes detailed requirements testing, most dirty testing, stress testing, and system testing areas such as performance, recovery, security, and configuration." [Bei90, p. 429]

"Stage 3: Enlightened Self-Testing – Some software development groups that have implemented a successful stage 2 process have returned system testing responsibility to the designers and done away with independent testing. This is not a reactionary move but an enlightened move. It has worked in organizations where the ideas and techniques of testing and software quality assurance have become so thoroughly embedded in the culture that they have become unquestioned software development paradigms. The typical prerequisites for enlightened self-testing are 5-10 years of effective independent testing with a continual circulation of personnel between design and test responsibility. Enlightened self-testing may superficially appear to be the same as the preconscious stage, but it is not. Component testing is as good as it can be, a metric-driven process, and real quality assurance (as distinct from mere words) is integral to the culture. Attempts, such as they are, to leapfrog stage 2 and go directly from stage 1 to stage 3 have usually been disastrous: the result is a stage 1 process embedded in stage 2 buzzwords. It seems that stage 2 is an essential step to achieve the quality acculturation prerequisite to a stage 3 process." [Bei90, p. 429-430]

"Beyond Stage 3 – As you might guess, some successful stage 3 groups are looking beyond it and a return to independent testing. Are we doomed to a cyclic process treadmill? Probably not. What we're seeing is a reaction to an imperfectly understood process. Eventually, we will learn how to relate the characteristics of a software product and the developing culture to the most effective balance between enlightened self-testing and independent testing." [Bei90, p. 430]

4 TESTING STANDARDS AND PRACTICES

In this chapter, the testing standards and practices that closely relate to the development of BlueGiga's testing process are described. Among them are the IEEE Standards for Software Unit Testing and Software Test Documentation. They represent the waterfall approach to software development and testing.

The process of Bluetooth testing and qualification is described here, as is the TTCN language used by qualified test facilities. A brief introduction to usage-based system testing is also given.

4.1 IEEE Standards

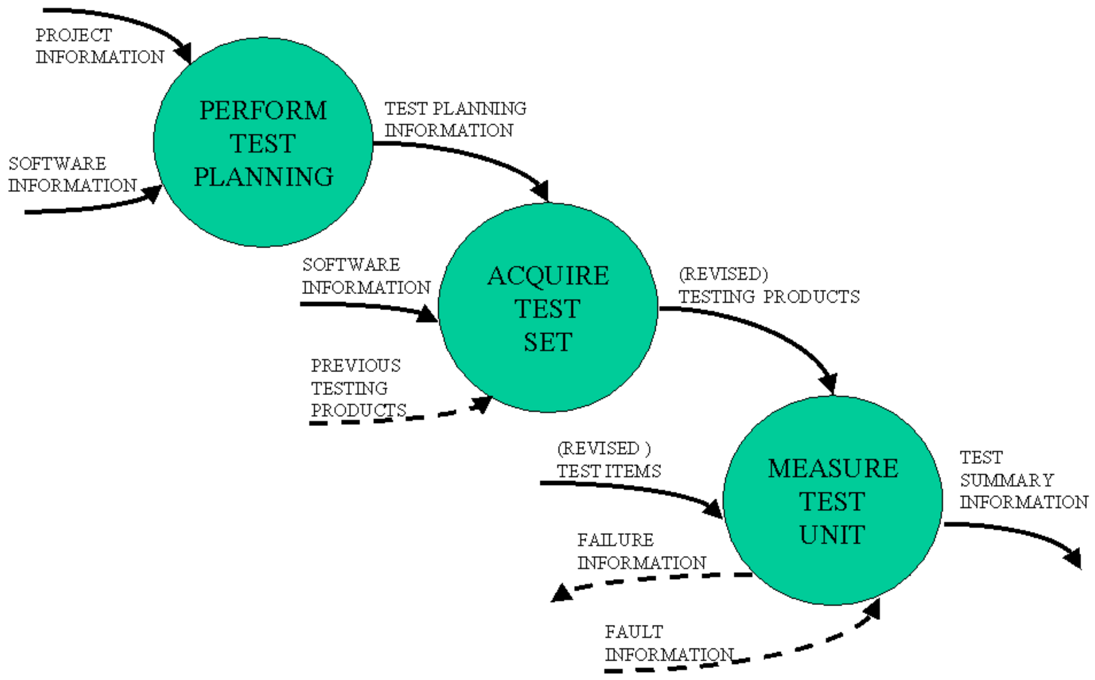
According to IEEE Standard for Software Unit Testing the unit testing process is composed of three phases that are partitioned into a total of eight basic activities [IEEE86, p. 3-4]. The activities are described in Table 1. From IEEE Std. 829-1983. Copyright 1986 IEEE. All rights reserved.

Table 1. Unit Testing Process. (Copyright© 1986 IEEE)

Unit testing process	
Phases	**Activities**
Perform the test planning	Plan the general approach, resources and schedule
	Determine features to be tested
	Refine the general plan
Acquire the test set	Design the set of tests
	Implement the refined plan and design
Measure the test unit	Execute the test procedures
	Check for termination
	Evaluate the test effort and unit

The major dataflows into and out of the phases are shown in Figure 3.

Figure 3. Major Dataflows of the Software Unit Testing Phases. (Copyright© 1986 IEEE)

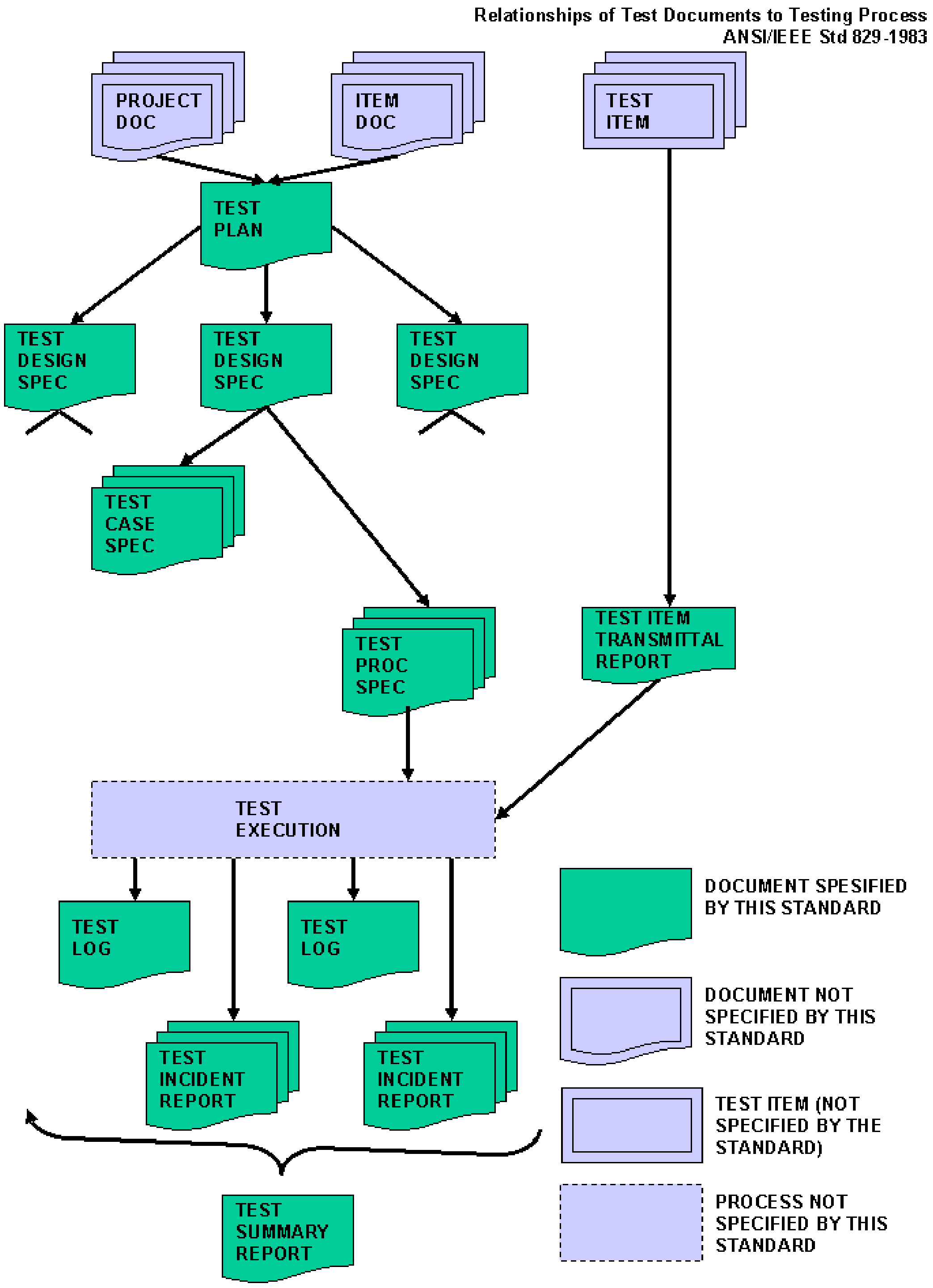


(Source: ANSI/IEEE Std 1008-1987, "IEEE Standard for Software Unit Testing". Permission to reprint obtained from IEEE.)

As can be seen from Figure 3, a critical factor for the success of a testing process is the availability of sufficient project and software information. Software information consists of requirements, design and implementation information. The need for other than requirements information arises from the fact that even though testing measures actual behavior against required behavior it is not usually feasible to test all possible situations. In addition, requirements do not often provide sufficient guidance in identifying situations that have high failure potential.[IEEE86, p. 19] From IEEE Std. 829-1983. Copyright 1986 IEEE. All rights reserved.

The basic software documents that were used in the documentation of the bit error rate test discussed in chapter 7.1 are outlined in the IEEE Standard for Software Test Documentation. The documents, presented in Figure 4, cover test planning, test specification, and test reporting [IEEE83, p. 4].

Figure 4. Relationships of Test Documents to Testing Process. (Copyright© 1991 IEEE)


Relationships of Test Documents to Testing Process
ANSI/IEEE Std 829-1983
PROJECT DOC
ITEM DOC
TEST ITEM
TEST PLAN
TEST DESIGN SPEC
TEST DESIGN SPEC
TEST DESIGN SPEC
TEST CASE SPEC
TEST PROC SPEC
TEST ITEM TRANSMITTAL REPORT
TEST EXECUTION
TEST LOG
TEST LOG
TEST INCIDENT REPORT
TEST INCIDENT REPORT
TEST SUMMARY REPORT
DOCUMENT SPESIFIED BY THIS STANDARD
DOCUMENT NOT SPECIFIED BY THIS STANDARD
TEST ITEM (NOT SPECIFIED BY THE STANDARD)
PROCESS NOT SPECIFIED BY THIS STANDARD

The **test plan** prescribes the scope, approach, resources, and schedule of the testing activities. It identifies the items to be tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan. [IEEE83, p. 10] From IEEE Std. 829-1983. Copyright 1991 IEEE. All rights reserved. In summary, it contains the information what is to be tested plus the resource allocation, time table and risk analysis information. To put it as simply as possible, test plan is a managerial level plan for the testing.

Test specification is covered by three document types: a test design specification, a test case specification and a test procedure specification [IEEE83, p. 3]. “A **test design specification** refines the test approach and identifies the features covered by the design and its associated tests. It also identifies the test cases and test procedures, if any required to accomplish the testing and specifies the feature pass/fail criteria.” [IEEE83, p. 3] From IEEE Std. 829-1983. Copyright 1991 IEEE. All rights reserved. One could use only the test design specification if the details separated from the test design to test case and test procedure specifications were also defined in the test design specification. The approach of using only the test design specification, together with the test summary report, is actually suggested in the IEEE Standard for Software Unit Testing [IEEE86, p. 8] as the minimum requirement for conforming to that specification. From IEEE Std. 829-1983. Copyright 1986 IEEE. All rights reserved. If the test case and test procedure specifications are used, the most important content of the test design specification is the feature pass/fail criteria.

“A **test case specification** documents the actual values used for input along with the anticipated outputs. A test case also identifies any constraints on the test procedures resulting from the use of that specific test case. Test cases are separated from test designs to allow for use in more than one design and to allow for reuse in other situations.” [IEEE83, p. 3] From IEEE Std. 829-1983. Copyright 1991 IEEE. All rights reserved. In summary, a test case is a set of inputs and their expected outputs.

“A **test procedure specification** identifies all steps required to operate the system and exercise the specified test cases in order to implement the associated test design. Test procedures are separated from test design specifications as they are intended to be followed step by step and should have no extraneous detail.” [IEEE83, p. 3] From IEEE Std. 829-1983. Copyright 1991 IEEE. All rights reserved. Aka, the test procedure specification is the user manual for the tester.

“Test reporting is covered by four document types: a test item transmittal report, a test log, a test incident report and a test summary report. A **test item transmittal report** identifies the test items being transmitted for the testing in the event that separate development and test groups are involved or in the event that a formal beginning of the test execution is desired. A **test log** is used by the test team to record what occurred during the test execution. A **test incident report** describes any event that occurs during the test execution which requires further investigation. A **test summary report** summarizes the testing activities associated with one or more test design specification.” [IEEE83, p. 3] From IEEE Std. 829-1983. Copyright 1991 IEEE. All rights reserved. In addition, a test summary report evaluates the test items based on the test results. A test summary report also comments on the comprehensiveness of the testing.

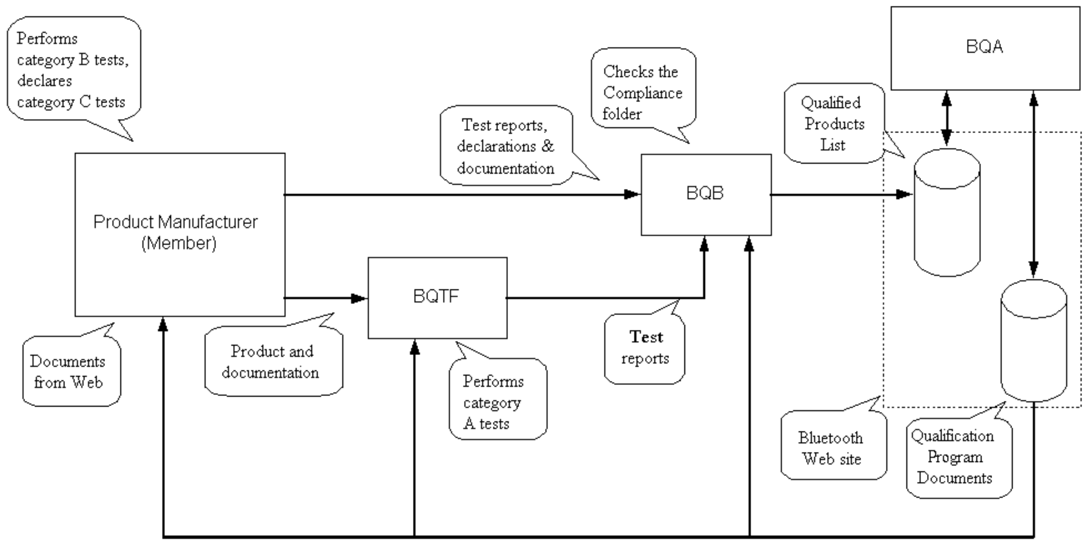
4.2 Bluetooth Testing and Qualification

“Bluetooth qualification sets some minimal testing standards for all products which use Bluetooth wireless technology. Qualification is a necessary precondition for the intellectual property license for Bluetooth wireless technology. Qualification is also necessary in order to apply applicable Bluetooth trademark to a product. However, neither the trademark nor Bluetooth qualification guarantee that a product fully complies with the Bluetooth specification. That remains the responsibility of the product manufacturer." [Blu01a]

4.2.1 The Process

The Bluetooth Qualification process, in Figure 5, is explained on the Bluetooth web page [Blu01a] and more specifically in the Bluetooth Qualification Program Reference Document (BQ PRD) [Blu01d]. An organization wanting to qualify a product has to become a Bluetooth Special Interest Group (SIG) Member before the product can be qualified. Normally this happens at the beginning of the product development before the qualification process starts.

Figure 5. Qualification Process [Blu01a, Blu01d, p. 33].


Performs category B tests, declares category C tests
Test reports, declarations & documentation
Checks the Compliance folder
Qualified Products List
BQA
Product Manufacturer (Member)
BQB
BQTF
Documents from Web
Product and documentation
Performs category A tests
Test reports
Bluetooth Web site
Qualification Program Documents

Before explaining the process, it is necessary to go through some essential definitions. The Bluetooth Qualification Review Board (BQRB) is responsible for managing, reviewing and improving the Bluetooth Qualification Program. The Bluetooth SIG promoter companies appoint delegates to the BQRB. The Bluetooth Qualification Administrator (BQA) is responsible for administering the Bluetooth Qualification Program on behalf of the BQRB. The BQB (Bluetooth Qualification Body) is an individual person authorized by the BQRB to add products to the Qualified Products List. A Bluetooth Qualification Test Facility (BQTF) is a facility accredited by the BQRB to perform test cases requiring special capabilities. [Blu01d, p. 10]

Only a recognized BQTF may perform category A tests. The product manufacturer performs category B and C tests. Different test categories are explained in Table 2. [Blu01d, p.35-36] The product manufacturer can download information from the web. This information includes both development and qualification related material. Bluetooth Core and Profile Specifications are needed to develop a Bluetooth product. Information, instructions and templates on the qualification is provided in the PRD (Program Reference Document), Test Specifications, Test Case Reference List, ICS/IXIT Proforma and Test Case Mapping Table. [Blu01d, p. 32]

"In many ways, the core specification is ambiguous. The test documents are far more rigorous, and its the test documents which determine whether a product will qualify to use the Bluetooth wireless technology and use the Bluetooth brand. Therefore, anyone planning on producing a Bluetooth product should familiarize themselves with the Bluetooth Qualification process and the Bluetooth Test Specifications, as well as the Core Specification." [Bra01, p. 392]

The Member, i.e. the product manufacturer, develops the product and selects a BQB to provide advice and assistance during the qualification process. The Member submits material necessary for the Compliance Folder to the BQB. [Blu01d, p. 32]

The Compliance Folder includes a description of the product, test plan, Implementation Conformance Statement (ICS), Implementation Extra Information for Testing (IXIT), Declaration of Compliance (DoC) and test reports. Description of the product includes identifying information and technical documentation. [Blu01d, p. 33-35]

With advice from the BQB, the Member will generate a product test plan, detailing all required testing for product qualification. If the test plan dictates category A testing, the product test plan will be used to co-ordinate the BQTF test efforts. The Member may also request a BQTF to perform category B tests and use the product test plan to co-ordinate the test efforts. The Implementation Conformance Statement (ICS) is used with test case mapping tables to determine which test cases are applicable for a product. The Test Case Reference List (TCRL) shows the category for each test case. The categorization of test cases, shown in Table 2, is the key to where test cases should be performed as well as the type of evidence that is required. [Blu01d, p. 34]

To evaluate a particular implementation, it is necessary to have a statement of capabilities and profiles, which have been implemented for a specific product. This statement is called an Implementation Conformance Statement (ICS). The Member, or the BQB based on input from the Member, prepares an ICS for use by the BQB and BQTF. [Blu01d, p. 34]

The Implementation Extra Information for Testing (IXIT) provides information related to the Implementation Under Test (IUT) and its testing environment which is required to be able to run the appropriate test suite. The answers to IXIT questions are used for testing purposes. [Blu01d, p. 34]

The Member must submit to the BQB the Declaration of Compliance (DoC). The DoC shall identify the specific product to be listed, including the hardware and software version numbers. The DoC proforma is available on the Bluetooth web site. [Blu01d, p. 35]

A Test report is required for all category A and B test cases. The report is necessary to demonstrate evidence of test results and to justify that all interoperability and conformance requirements are fulfilled. [Blu01d, p. 35]

After receiving the Compliance Folder or creating it based on input from the Member the BQB decides if the product is certified. The BQB lists the certified product on the Qualified Products List on the Bluetooth web page. If the product fails to qualify it can re-apply for qualification.

The BQ PRD (Bluetooth Qualification Program Reference Document) defines four categories of tests for Bluetooth products [Blu01d, p. 35]. The test categories are described in Table 2.

Table 2. Test Categories [Blu01d, p. 35-36].

Category A
Short name: Mandatory at an accredited BQTF.
Description: This test case is fully validated and commercially available in at least one implementation. The test case is mandatory and has to be performed at an accredited BQTF.
Category B
Short name: Declaration with evidence.
Description: The test case is mandatory and shall be performed by the Member. The Member declares that the IUT design meets the test case's conformance and interoperability requirements and justifies this declaration by reporting the testing results and the test set-up to the BQB. If the member does not follow the instructions in the test specification, it must specify how the test was performed.
Category C
Short name: Declaration without submittal of evidence.
Description: The test case is mandatory and shall be performed by the Member. The Member declares that the IUT test design meets the test cases conformance and interoperability requirements, and that the IUT has successfully passed the test case. No evidence is required to be submitted to the BQB.
Category D
Short name: Informative.
Description: A preliminary test case, not required for qualification. This category informs a Member of a test case which may be elevated later to a higher category. When appropriate, the Member is encouraged to perform these test cases.

A summary of the roles of different participants in the testing process is presented in Table 3 below. It shows that the entire engineering testing is the product manufacturer's responsibility. Also, the testing of interoperability, for example at the unplugfests, is the manufacturer's responsibility.

Table 3. Bluetooth Testing Process [Rob01, p. 14].

Testing process			
	Pretest	**BQB Testing**	**Post-testing**
Product manufacturer's role	Product development Engineering testing	Conduct category B, C, D tests * Generate test reports Build compliance folder	Provide compliance declaration Pay listing fee Option of more tests at unplugfests

BQB role	Assist with planning Qualification review requirement	Build compliance folder	Prepare and certify qualified products Formally list the product
BQTF	May assist with the above	Conduct category A tests	Not required
Explanation of categories: category A=validated and commercially available product tests to be performed by BQTF; B=manufacturer test with declaration and evidence; C=manufacturer test with evidence; D=preliminary informative test with no qualification value			

4.2.2 *Types of Testing*

"The testing strategies are different for RF, protocol conformance, profile conformance and profile interoperability tests [Blu01d, p. 39]." The RF, protocols and profiles are all tested for conformance. The profiles, and, at the moment, also the protocols are tested for interoperability. [Blu01d, Bra01]

"Conformance testing of a Bluetooth Product is defined as testing according to the applicable procedures given in the Bluetooth RF and protocol test specifications and the Bluetooth profile conformance test specification when tested against a reference test system. The objective of the RF, protocol and profile test specifications is to provide a basis for conformance tests for Bluetooth devices giving a high probability of compliance with the Bluetooth System Specifications. Conformance tests are performed by an accredited BQTF or the Member according to the test case category." [Blu01d, p. 40]

"Interoperability testing is defined as a functional testing performed according to the applicable instructions/guidelines given in the Bluetooth Profile Interoperability Test Specification against another operational Bluetooth product [Blu01d, p. 40]." "Profile interoperability testing helps to determine that products supporting the same profile actually interoperate as intended (and as specified). Interoperability tests may uncover the unique problems that become evident when actually communicating between products, especially when the products are produced by different manufacturers." [Blu01d, p. 40]

To strengthen confidence in lower layer interoperability protocol interoperability testing is done in the initial phase. This is performed using designated protocol test products commonly referred to as 'Blue Units'. [Blu01d, p. 40]

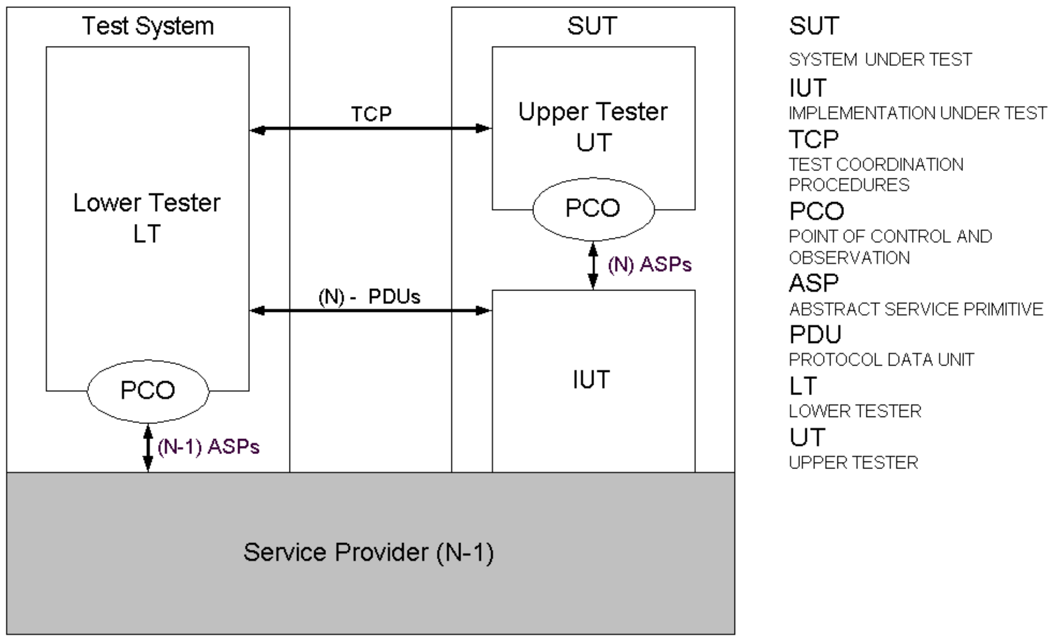
4.3 Tree and Tabular Combined Notation

"Tree and Tabular Combined Notation (TTCN) is a standardized language for describing black-box testing for reactive systems such as communication protocols and services [Wil00]." TTCN is independent of test methods, layers and protocols [ISO98, p. xi].

4.3.1 *Abstract Conformance Test Architecture*

Conformance means, in this context, that a product has met the requirements of a relevant specification. [TL99, glossary, p. 1] Conformance testing is done against a test system. The test system is divided into two parts: the lower tester (LT) and the upper tester (UT). The test system architecture is presented in Figure 6.

Figure 6. Abstract Conformance Test Architecture [Juk99].



The lower tester's (LT) lower interface connects to the (N-1) –service provider, through which it indirectly controls and observes the abstract service primitives (ASPs). The connection between the lower tester and its environment is called a Point of Control and Observation (PCO). The traffic that the test system can control and observe flows through the PCO. Because there exists a service provider between the LT and the implementation under test (IUT), the (N-1) –service provider has to be reliable – the object of testing is to test the IUT, not the IUT and the service provider together. [Hak94, p. 12-15]

The UT communicates with the IUT through the IUT's upper interface via another PCO using N –primitives. The test method is distributed, because the UT is located logically to the system under test (SUT). The UT does not necessarily have to be a separate entity, it can also be a part of the system under test. The UT can communicate with test system using test coordination procedures (TCPs). [Juk99, p. 10]

The terms N-1 and N refer to the OSI model layers of a protocol stack. The IUT is seen as the N-layer of the protocol stack, which it often, but not necessarily always, is. The OSI model consists of several protocol layers. Layers that are on top of each other communicate through service access points (SAPs). Peer protocol layers communicate with each other using protocol data units (PDUs). A picture of the OSI model is provided in appendix 1.

4.3.2 ASN.1 and TTCN Type Definitions

TTCN contains as a sublanguage ASN.1 (Abstract Syntax Notation One), which is used in the definition of data types by several telecommunication protocols. The definitions of data types can thus be reused in testing. A more important benefit of ASN.1 is that compilers and tools have been developed for it. This makes the compiling of the abstract test data easier. [Juk99, p. 13]

ASN.1 contains a selection of pre-defined primitive types of which the user can compose more complicated structured types. There is also the possibility to subtype, e.g. to limit the value range of primitive types or the size of structured types. [Juk99, p. 13]

4.3.3 TTCN Test Suite

The TTCN test suite consists of two parts. Test Suite Overview and Declarations Part form the body of the test suite by describing the whole test suite. Constraints Part and Dynamic Part depend on the test case. The test suite is formulated by filling in the tables, called proformas, in these parts. [ISO98, p. 15-16; Juk99, p. 15] There would also be an Import Part, if the test suite used objects defined in other test suites. [ISO98, p. 16]

4.3.3.1 Suite Overview

The Test Suite Overview briefly describes the overall purpose of the testing. It presents the test cases and test steps. It also tells to which group each test case belongs. This information helps in documenting and managing the test suite. [ISO98, p. 16-22; Juk99, p.16]

4.3.3.2 Declarations Part

In the Declarations Part, the data types and the abstract service primitives (ASPs) consisting of them, protocol data units (PDUs), parameters that transmit values from the environment when the test suite is run, constants and the variables that temporarily store the values of the fields of received messages are defined. Also, the timers and the test architecture are defined here. The definition of the points of control and observation (PCOs) is essential to the test architecture. Messages are sent and received at the PCOs. PCO’s type tells if it is part of the upper or lower tester. [ISO98, p. 24-73; Juk99, p. 16]

4.3.3.3 Constraints Part

In the Constraints Part, the values of the ASP parameters and PDU fields that are to be sent or received by the test system are defined. The dynamic behavior description references constraints to construct outgoing ASPs and/or PDUs in SEND events; and to specify the expected contents of incoming ASPs and/or PDUs in RECEIVE events. [ISO98, p. 73] Values of all TTCN and ASN.1 types can be used in constraints. Constraints can be parameterized to increase modularity of the test case. [ISO98, p. 74]

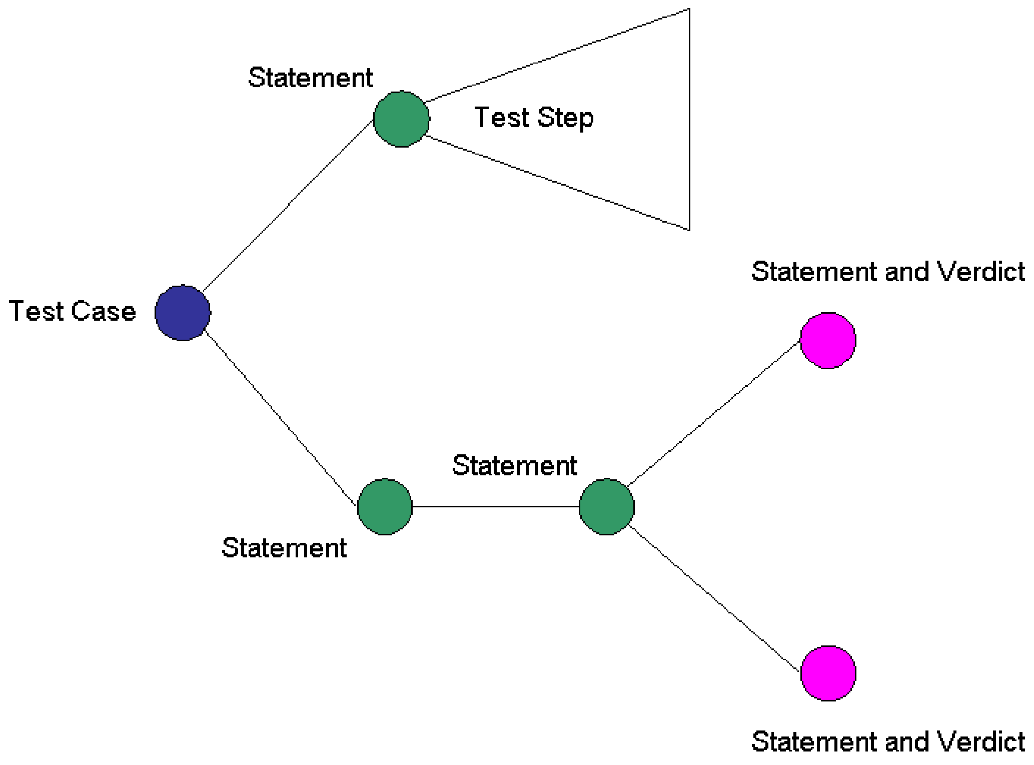
4.3.3.4 Dynamic Part

The behavior and functionality of the test suite is described in the dynamic part with three different kind of tables. The difference between these tables is in the header. The format of the body is the same for all three. These tables are each used in a different way. [Cet01, p. 41]

A test case consists of the test events that relate to the testing of one logical function. The event may be the sending or receiving of a message, or something else like setting the value of a variable. This is described in the Test Case Behavior table. A recurring test event or a series of test events can be defined as a test step in a Test Step Behavior table. Test steps can be parameterized like constraints. Parameterization makes the reuse of test steps easier. The third table is the Default Behavior, which defines the default behavior of a test case or a test step. [Juk99, p. 18]

The test case can be presented as a tree structure as in Figure 7. The test case describes both the chronological order of the events and the result of the test at the end of a series of test events. In a TTCN table, the tree structure is presented with indentations. The sequential events in the test case advance in a chronological order from one row to another, top to bottom. Every new event is indented one indentation level to the right. [Juk99, p. 19] An example of a TTCN Test Case Dynamic Behavior table is provided at page 60.

Figure 7. Test Case Behavior [ISO98, p. 94].



When the tree branches the test case can advance into alternate directions. At this point, you compare the type and contents of the received message row by row to the type and constraint of each alternative and to the truth value of the possible conditional statement, and the execution advances to the branch of the first applicable alternative. In the table, alternate events are at the same indentation level. After the alternative has been executed, the execution continues from the next row after the group of alternatives, one indentation level to the right. [Juk99, p. 19]

There are two mechanisms in TTCN that provide assignment of verdicts to a test case: preliminary results and explicit final verdicts. TTCN has a predefined test case variable called R. This variable may be used in expressions as a read-only variable and in the verdict column of a behavior description. It is used to store preliminary results. Changes are made to its value by entries in the verdict column. It may only take one of the values: pass, fail, inconclusive or none. Preliminary results are marked with the value of R in parenthesis in the verdict column, a final verdict is not in parenthesis. [Cet01, p. 71-74]

Execution of a test case is terminated either by reaching a leaf of the test case behavior or an explicit final verdict on the behavior line. A final verdict may be one of the verdicts defined in Table 4. If no explicit final verdict is reached, the final verdict is the value of R. If R is still bound to none, then there is a test case error. [Cet01, p. 74]

Table 4. Verdicts [Cet01, p. 73-74].

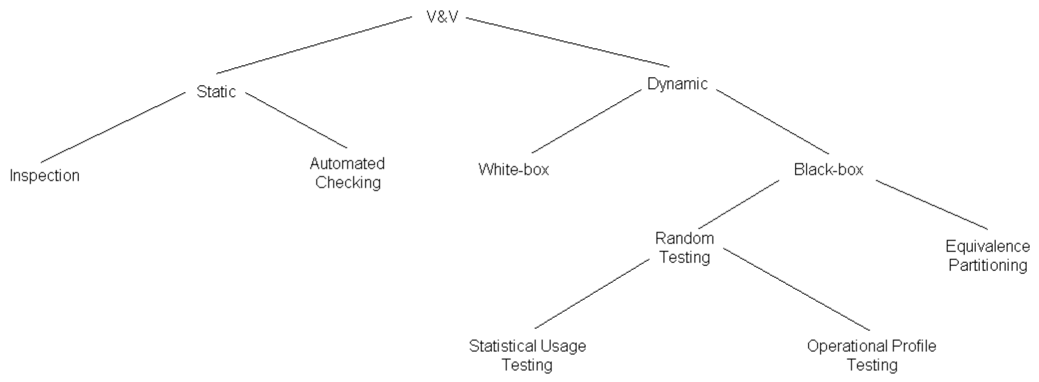
Verdict	**Meaning**
P or PASS	the test purpose has been achieved
I or INCONC	something has occurred which makes the test case inconclusive
F or FAIL	a protocol error has occurred
R	no explicit final verdict is reached

In a test case, one must take into account all the test events that may possibly occur. On the other hand, only a small subset of all possible combinations is interesting, and it is not even possible to write TTCN events for all possible combinations of events. When an event that is not defined occurs, the execution continues to a series of test events defined in the Default table. The Default step is automatically extended to the level of each set of possible test event as the last alternate event. An execution of a test case that ends in the Default step is automatically failed. [Juk99, p. 19]

4.4 Usage-Based Testing

“Verification & validation strategies may be divided into two main classes: (1) static V&V including V&V methods that do not execute the artifact under scrutiny, and (2) dynamic V&V including methods that exercise a software artifact by executing it with sample input data. These two classes can be broken down further into different types of methods. Figure 8 shows a partial classification of V&V methods.” [Reg99, p. 16]

Figure 8. Classification of Verification & Validation Techniques [Reg99, p. 16].

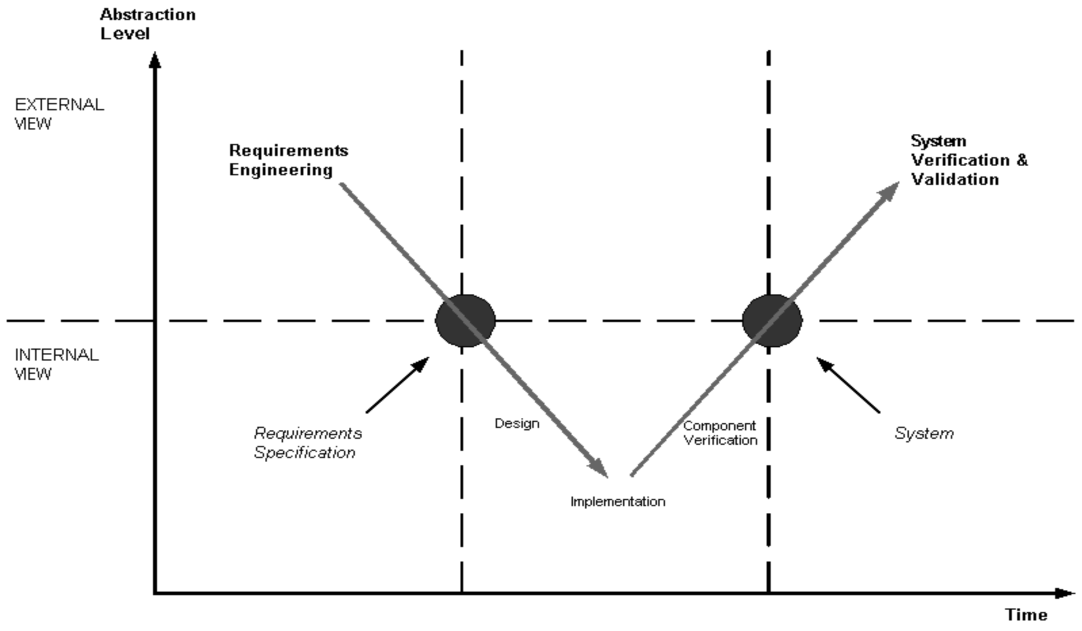


“Both Requirements Engineering (RE) and Verification & Validation (V&V) view the system under development at a higher abstraction level compared to design and implementation, and both disciplines have external view of the system (see Figure 9), where the system usage is in focus, rather than its internal structure. In both the RE and V&V research communities, there exist concepts related to system usage, namely the *use case* concept in RE and the concept of *usage testing* in V&V.” [Reg99, p. 17]

“There is an intimate relation between requirements specification and system validation; the major goal of validation is to show, for example through testing, that a system correctly fulfils its requirements [Reg99, p. 87].” “In black-box techniques test cases are derived from a system specification, and hence have a natural connection to requirements [Reg99, p. 17].”

“The reliability of a system depends not only on the number of defects in a software product, but also on how it is executed in operation. This implies that reliability testing must resemble operational usage, i.e. test cases are selected according to a usage profile.” [Reg99, p. 17]

Figure 9. External and Internal Views in the Software Development Process [Reg99, p. 88].



An interesting research question formulated by Regnell [Reg99, p. 19] is: "How can use case modeling be integrated with system testing, so that the usage information contained in use case models can be utilized for test case creation and assessment of reliability?" Nearly all developers mention the need to base system tests on scenarios, but current (1999) practice rarely satisfies this demand as most projects lack a systematic approach for defining test cases based on scenarios [Reg99, p. 25].

Regnell proposes two alternatives for integration of use case modeling and usage-based testing, the transformation and extension approaches. These methods are tried out in the experimental part of this thesis. Therefore, the concepts of use cases, operational profile and the transformation of use cases into an operational profile using the different approaches are discussed here.

4.4.1 *Use Case*

The major activities in creating a use case model according to an extension of Object-Oriented Software Engineering (OOSE) approach used by Regnell are:

- Elicit actors and their goals.
- Define use cases based on the actors and their goals.
- Elicit scenarios for each use case.
- Describe the events in the scenarios. [Reg99, p. 122]

"The main concepts in these activities are briefly described in the following. Users can be of different types, called **actors**." [Reg99, p. 122] "Each actor defines a particular role [Sch98, p. 12]." "One physical person may be represented by several actors because that person takes on different roles with regard to the system. Or several physical people might be represented by one actor because they all take on the same role with regard to the system." [Sch98, p. 12] "A user is thus an instance of an actor [Reg99, p. 122]."

Each actor has a set of **goals**, reflecting common characteristic [Reg99, p. 122]. "Goals are objectives that users have when using the services of a target system. Thus, goals are used to categorize users into actors." [Reg99, p. 122]

"A **service** is a package of functional entities (features) offered to the users in order to satisfy one or more goals that the users have [Reg99, p.122]." "A **use case** represents a usage situation where one or more services of the target system are used by one or more actors with the aim to accomplish one or more goals [Reg99, p. 122]."

"If you pick one particular path through the use case, that is called a scenario [Sch98, p. 30]." "A scenario may either model a successful or an unsuccessful accomplishment of one or more goals. A use case may cover an unlimited number of scenarios as it may include alternatives and repetitions. A scenario, however, is a specific and bound realization of a use case, with all choices determined to one specific path." [Reg99, p. 122-123]

"When describing the events of each scenario it is useful to define a **data model**, **messages** of the system and **system actions**. The latter mean system intrinsic events which are atomic in the sense that there is no communication between the target system and the users that participate in the use case." [Reg99, p. 122, 124]

4.4.2 *Operational Profile*

A method for deriving an operational profile used by Regnell "is summarized below in five steps:

- Develop a customer type list.

- Develop a user type list.
- List system modes.
- Develop a functional profile.
- Convert the functional profile into an operational profile." [Reg99, p. 127]

"The **customer type list** collects the different types of customers that *acquire* the system. A customer type represents a set of customers which utilize the system in a similar manner." [Reg99, p. 127] "The **user type list** collects the different types of users that *use* the system. This list is not necessarily the same as the customer type list. For larger systems, it is generally not the same." [Reg99, p. 127]

"System modes represent a set of functions or operations that are grouped together in a way that is suitable for the application. The system modes need not to be orthogonal to each other; a function or operation can be member of different system modes." [Reg99, p. 127] Criteria for defining system modes can be: Relatedness of functions/operations to larger task, significant environmental conditions, operational architectural structure, criticality, customer or user, and user experience [Reg99, p. 127].

"The first step in defining the functional profile is to create a **function list**. Functions are defined from the user's perspective and do not involve architectural or design factors." [Reg99, p. 127] The function list can be modified by taking environmental variables into account [Reg99, p. 128].

"Now the profile is attached to the functions. We choose an implicit form with **key input variables** for each function. The variables decide the variants of the functions. The different values of the variables are called levels." [Reg99, p. 128] Let's take, for example, a light tuning system and a function called *set lights on*. The key input variable for the function is the *status of the lights* after the operation. The levels of the key input variable are *success* (lights on), *no current* (from the system's point of view the lights are switched on, but there is no current), and *failure* (lights off).

"Finally, the functions are mapped onto **operations** [Reg99, p. 128]." Let's say that the light tuning system uses Bluetooth to connect from a PDA to the light switch. Then, for example, the *set lights on* function can be built up by three operations, *establish Bluetooth link, set lights on*, and *close Bluetooth link*. "It can be noted that operations may be involved in performing different functions [Reg99, p. 128]." For example, the operations of establishing and closing the Bluetooth link are also involved in function *set lights off*.

4.4.3 Derivation of an Operational Profile using Transformation Approach

"A use case model and operational profile model have very much information in common [Reg99, p. 131]." "The operational profile model primarily adds the profile information i.e. the probabilities for use of different system capabilities. However the models originate from different disciplines and different terminology is used." [Reg99, p. 131]

"The transformation approach takes the information in the use case model and builds the operational profile model based on that information and additional information from other sources. In order to find the common parts in the two models the concepts in the two domains are elaborated below." [Reg99, p. 131] "The requirements model does not include quantitative aspects on usage frequencies, while the operational profile model does. This information has hence to be derived in addition to the use case information." [Reg99, p. 131]

"The **customer types** in the operational profile model address the customers, which acquire the system, and the probability information for different types of customers. If high-level goals are included in the use case model, customers might be found among the **actors**, otherwise other sources have to be consulted." [Reg99, p. 131]
"The **user types** in the operational profile generally correspond to the actors in the use case model. Both concepts refer to categories of objects, either human users or other systems that interact with the system." [Reg99, p. 132] "The **system mode** is a set of functions and operations that relate to each other. This corresponds in part to the **service** concept of the use case model. However, the system mode may also cover profiling information, and that part cannot be found in the use case model." [Reg99, p. 132]

"The usage of **functions** in the operational profile model is described by **use cases** in the requirements model. Variants of functions in the operational profile are distinguished by **key input variables**. Depending on the value of the variable (the level), different alternatives of the function are chosen. The levels of the key input variables have their counterparts in the **scenarios** of the use case model, which are variants of a use case. A specific combination of key input variables gives a specific scenario, i.e. a realization of a use case." [Reg99, p. 132]

"In the operational profile model, **operations** are tasks accomplished by the system in order to deliver a function to the user. A similar concept in the use case model are the **system actions**." [Reg99, p. 132]

The mapping between concepts in an operational profile model and a use case model are summarized in Table 5.

Table 5. Mapping of Terminology [Reg99, p. 132].

Operational profile model		**Use case model**
Customer type	≈	Actor
User type	=	Actor
System mode	≈	Service
Function	≈	Use case
Key input variable	≈	Scenario

Operation	≈	System action

4.4.4 Derivation of an Operational Profile using Extension Approach

"Instead of creating a completely new model by transforming the use case model into a state hierarchy model, we can adopt the principal ideas behind statistical usage profiles and create an extended use case model, complemented with event statistics. If we can create well defined semantics for how test cases can be generated directly out of the use case model, we will save the effort of making two different models." [Reg99, p. 110]

"The basic idea behind the model extension is to complement every part of the use case model where there are non-deterministic choices with the probabilities of the different choices." [Reg99, p. 111]

"On the **environment level** , we have to extend the use case model with information on the number of probabilities for each actor and the probabilities for each actor to generate stimuli to the system. Furthermore, it has to be analyzed if there are variants of actors with respect to their usage profile." [Reg99, p. 111]

"On the **structure level** we have to add profile information to the scenarios [Reg99, p. 112]." "On the **event level**, probabilities are attached to each choice in the model. For example, the alternative operator introduces a non-deterministic choice between two or more alternatives." [Reg99, p. 112] If we decorate the alternative operator with probabilities, we can draw random numbers to decide which alternative is chosen during test case generation [Reg99, p. 112]. "This way we construct stochastic semantics for each operator that determines how to generate scenarios. The scenarios are then used as test cases." [Reg99, p. 112]

5 BLUETOOTH TECHNOLOGY

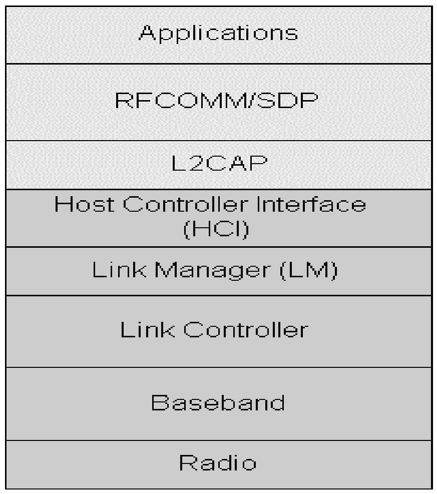
In the beginning, the aim of Bluetooth technology was to provide short range connectivity without cables. The devices where the cable was to be removed were, to mention only few, mobile phone handsets, headsets and portable computers. Since then it has extended to more imaginative application areas, such as the personal area network (PAN). The purpose of PAN is to allow ad-hoc networks to be set-up anywhere where there are Bluetooth enabled devices. The different applications are made possible through the use of Bluetooth profiles, which define how a particular application can be implemented and what parts of the protocol stack it should use. [Blu01a, Bra01]

The purpose of this chapter is to give the reader an overview on Bluetooth technology limited on those parts interesting from the point of view of the BlueGiga Technologies' WRAP (Wireless Remote Access Platform) product family. This is done by presenting the Bluetooth stack extremely concisely and those Bluetooth application profiles that are used in the WRAP product family. This should help the reader to understand the WRAP product family at a very general level.

5.1 Stack

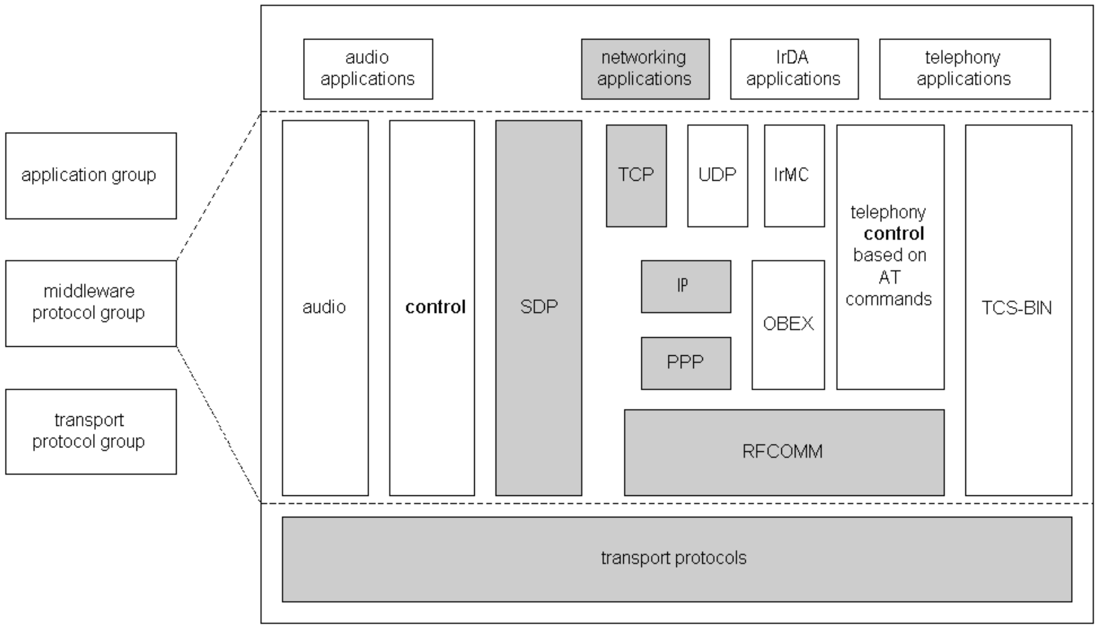
The Bluetooth stack, in Figure 10, has been divided into Bluetooth module and host. The module contains the lower parts from the radio up to the host interface. The host means the higher parts of the system which would typically exist on or in a host system that is Bluetooth enabled. [Bra01, xx] In Figure 10, baseband is presented as an independent protocol layer. In practice, when manufacturer's talk about baseband, they usually mean the baseband chip. The chip usually contains, in addition to the baseband, a link controller and a link manager.

Figure 10. The Bluetooth Stack [Bra01, p. 7].



In Figure 11, which is adapted from Bluetooth Revealed [Mil01, p. 67], the WRAP product family's position in the Bluetooth protocol stack is presented. The Bluetooth specific protocols implemented above transport protocol layer, e.g. above Logical Link Control and Adaptation Protocol (L2CAP), are RFCOMM and SDP. LAN Access Profile describes the use of Point-to-Point Protocol (PPP) over a Bluetooth link. PPP and TCP/IP (Transport Control Protocol/Internet Protocol) are not Bluetooth specific protocols.

Figure 11. WRAP's Position in the Protocol Stack.

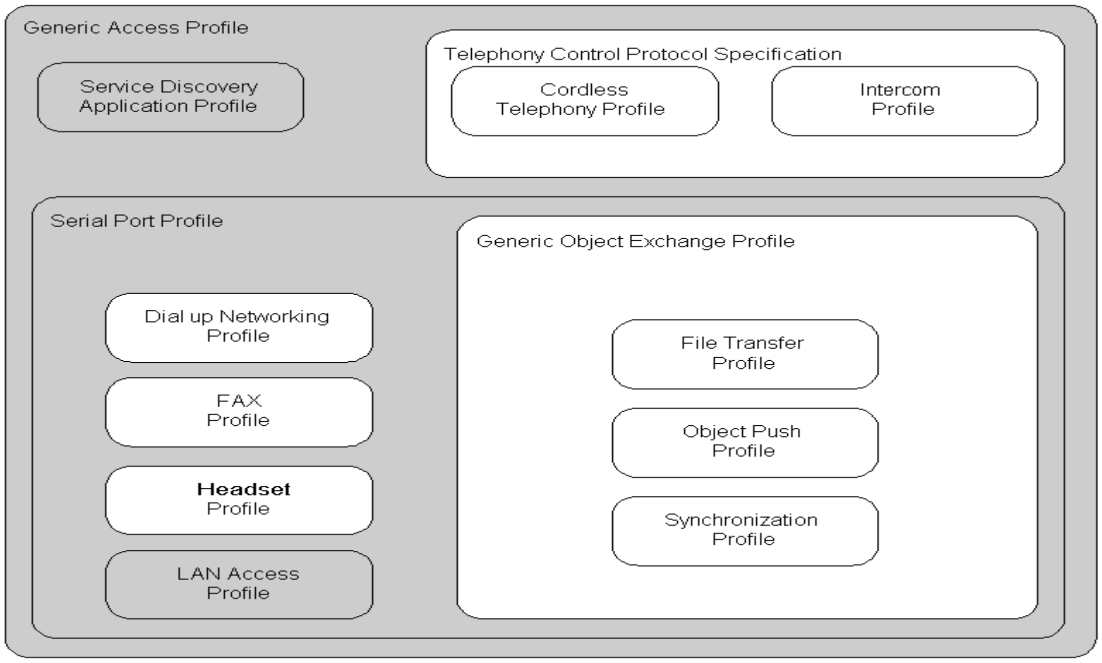


5.2 Profiles

As mentioned earlier, the WRAP implements LAN Access Profile for the creation of PPP connections over a Bluetooth link. Figure 12, adapted from Bluetooth, Connect without Cables [Bra01, p. 266], shows how

Bluetooth profiles are organized into groups. The profiles used in WRAP are colored gray. Each profile builds upon the one beneath and inherits features from below [Bra01, p. 266]. It can be seen that the LAN Access Profile builds upon Serial Port Profile. The Serial Port Profile builds upon Generic Access Profile as do all the other profiles. The Service Discovery Application profile is used to inquiry if the service required by a client is provided by a Bluetooth host.

Figure 12. Bluetooth Profiles Used in WRAP.



5.2.1 Generic Access Profile

“The Generic Access Profile or GAP, forms a common basis for all Bluetooth profiles and hence for the common interoperable usage of the group of Bluetooth transport protocols. One of its important contributions is its definition of a standard set of terminology.” [Mil01, p. 211] “Having this common vocabulary helps to remove ambiguity in all of the profiles, which is a key element of enabling interoperable implementations [Mil01, p. 211].”

“With this common terminology in place, most of the GAP is devoted to defining the procedures necessary to establish Bluetooth connections [Mil01, p. 211].” This includes device and name discovery, inquiry procedures, pairing and bonding, and link, channel and connection establishment [Mil01, p. 211]. “For all of these considerations, the GAP provides common and standard procedures, in some cases including flowcharts. The importance of defining the fundamental communication operations cannot be overstated: without well-defined, interoperable methods for basic communication between devices, none of the other profiles could be realized.” [Mil01, p. 211]

5.2.2 Service Discovery Application Profile

"The Service Discovery Protocol (SDP) provides means for an SDP client to access information about services offered by SDP servers [Bra01, p. 217]." The Service Discovery Application interacts with the SDP layer of the stack in the device where it resides to initiate SDP interactions with one or more SDP layers in other devices, so as to learn about services in those other devices [Mil01, p. 219]. In other words, the Service Discovery Application Profile (SDAP) specifies how the user-level applications, e.g. Service Discovery Applications, work together to support a usage scenario, which is retrieving information about services residing in other devices.

"A server can be any Bluetooth device offering a service which can be used by another Bluetooth device, and a client can be any device wanting to use a service. So, a device could be an SDP client and server at the same time." [Bra01, p. 217]

"SDP servers maintain a database of service records. Each service record provides information that a client needs to access a service. This information may include URLs for executables, documentation, and icons associated with the service, So, a client may have to follow these URLs and retrieve information from elsewhere to be able to use the service." [Bra01, p. 217]

"To use SDP, an L2CAP channel must be established between the SDP client and server. This channel has a protocol service multiplexor reserved for SDP, so that any device can easily connect to the SDP service on another device. After the SDP information has been retrieved from the server, the client must establish a separate connection to use the service (the connection used for SDP cannot be used for anything else)." [Bra01, p. 217]

"Services have Universally Unique Identifiers (UUIDs) which describe them. The services defined by the Bluetooth profiles have UUIDs assigned by the standard, but service providers can define their own services and assign their UUIDs to those services. The UUIDs are allocated by a method that guarantees they will not be duplicated, so there is no need for a central authority or a central database to allocate the UUIDs." [Bra01, p.217]

"An UUID can be sent in a message asking a server if it supports the service identified by the UUID. Alternatively, instead of asking for a specific service, SDP can provide a mechanism for organizing services in trees, along with messages for browsing through the trees to look for a service." [Bra01, p. 217]

"SDP does not define the applications needed to drive the service discovery process, nor does it define an interface to applications; this is left up to implementers. If required, it may be used alongside other service discovery methods which provide Application Programming Interfaces (APIs)." [Bra01, p. 217]

5.2.3 Serial Port Profile

"The Serial Port Profile (SPP) is a transport protocol profile that defines the fundamental operations necessary to establish RFCOMM communications between two peer devices. Such a link is required for many of the concrete usage scenario profiles. In this respect the SPP is somewhat like the GAP, in that it describes how to establish necessary communication links that are in turn needed by other profiles. The SPP serves as a ‘building block’.” [Mil01, p. 240]

“As an abstract profile, the SPP is more likely to be used by middleware than directly by applications. Bluetooth adaptation software might use the SPP to instantiate a virtual serial connection for an application that expects to use serial communications. The application need not know that the serial port is an emulated wireless one, so long as the emulation is sufficiently accurate.” [Mil01, p. 243]

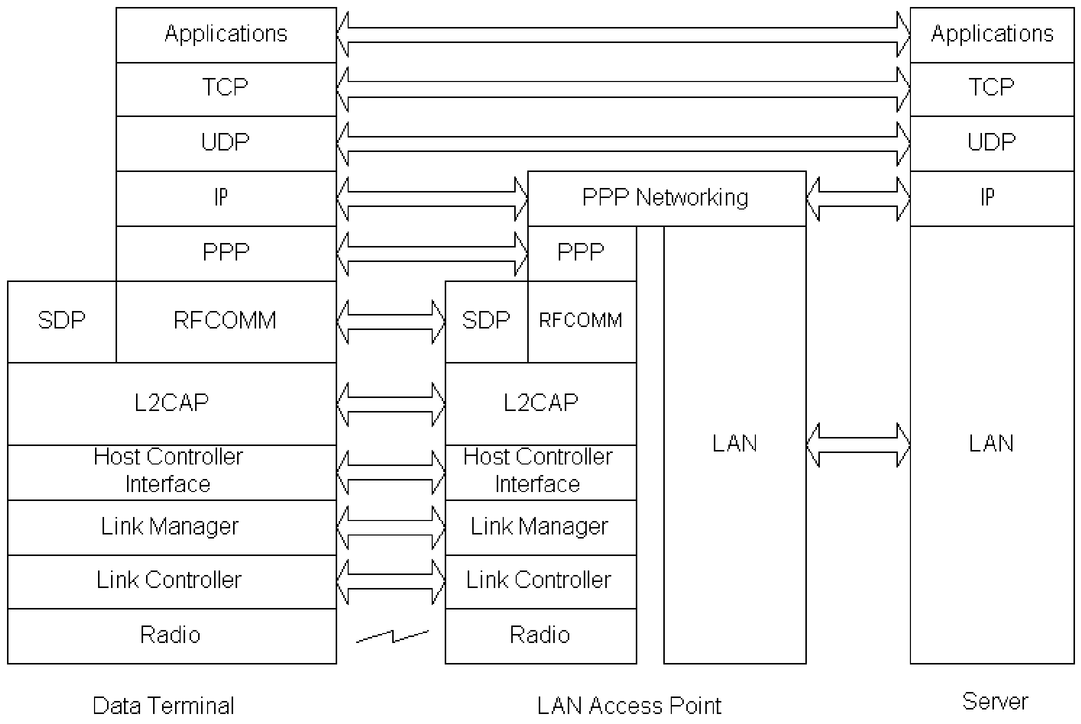
“Thus applications most likely will implement some other profile that makes use of the SPP [Mil01, p. 243]” – LAN Access Point in the case of WRAP product family. As Miller suggests in Bluetooth Revealed for profiles that make use of SPP, the SPP is supported “generically in the middleware” [Mil01, p. 243] in the WRAP product family. “An application follows the standard procedures” [Mil01, p. 243] for the Wireless Remote Access Platform (WRAP) “to open, configure, make use of and close the serial connection” [Mil01, p. 243]. The WRAP “middleware will transform the platform APIs into the corresponding functions of the SPP necessary to use RFCOMM over Bluetooth transports” [Mil01, p.243]. Thus the user, in this case the application programmer, needs only to know the commands used in WRAP platform API, knowledge of the specific Bluetooth commands is not necessary.

5.2.4 LAN Access Profile

“The LAN Access Profile allows a Bluetooth enabled device to access a fixed network via a Bluetooth link to a LAN Access Point (LAP) [Bra01, p. 277].” Such a device could be used in many scenarios: as a personal work area access point, replacing the network cable and allowing mobility within range of the access point; as a shared access point in a meeting room, allowing fast establishment of network connections; or as a public access point, allowing easy access to information and services, for example, at airport check-ins and waiting lounges. [Bra01, p. 277; Sai00, appendix II, p. 1]

“The LAN Access Profile specifies using PPP over RFCOMM to link an IP stack to the Bluetooth stack. The resultant configuration is shown in Figure 13. The figure shows a LAN Access Point acting as an intermediary between a Bluetooth device and a server resident on a LAN. However, the LAN Access Profile simply specifies how to layer an IP stack on top of a Bluetooth stack.” [Bra01, p. 277] “The most commonly described usage case for the LAN Access Profile” according to Miller [Mil01, p. 265] “is for a computer to access a LAN, although the usage case for the LAN Access Profile does enable the development of data access points that could be used simultaneously by multiple Bluetooth clients” [Mil01, p. 265] as is done in the WRAP product family’s Man-to-Machine use case. A specialized case of LAN Access Profile is its use to directly connect two devices for PPP communication between them rather than to access a larger network [Mil01].

Figure 13. Protocol Stack for LAN Access Profile [Bra01, p. 278].



“To reduce connection times the LAN Access Point’s address can be re-entered into connecting devices. The LAN Access Point periodically holds its links and page scans, allowing connecting devices to simply page the access point and conduct a Master/Slave switch to hand over control of the link to the LAN Access Point.” [Bra01, p. 277]

Peer-to-peer communication in purely ad-hoc, wireless networks is an area that is not yet mature. While many industry and academic efforts are underway, there is no robust, fully tested, widely accepted method for achieving it, especially one that is suitable for resource-constrained devices like many of those used in Bluetooth networks. Bluetooth PAN working group is specifying support for more general ad hoc peer-to-peer networking than what is currently supported in LAN access profile [Blu01a, Mil01, p. 282-283].

6 BLUEGIGA TECHNOLOGIES

“BlueGiga Technologies is an innovative wireless technology company based in Espoo & Turku, Finland and Stockholm, Sweden who develops and markets integrated product solutions and design services tailored to industrial customer’s needs. BlueGiga’s mission is to help the OEMs in creating wireless remote connectivity and pervasive access to electronic devices and systems in order to realize substantial cost savings and productivity improvements.” [BTG01c. p. 1] BlueGiga Technologies is an Adopter member of the Bluetooth SIG.

BlueGiga’s software development process conforms to the Boehm’s spiral model. Some practices common in extreme programming are also in use. Use cases are written at the requirements engineering phase.

“BlueGiga's Wireless Remote Access Platform is a pre-certified, fully integrated hardware and software solution. It combines standard networking and Internet technologies and the Bluetooth™ wireless technology in a turn-key platform enabling remote access to devices and systems. WRAP™ products can intelligently and independently run all needed protocols and application software without the need for a host processor. It is designed to significantly lower the investment and "time-to-market" for OEMs in developing solutions for new wireless applications or adding wireless and network connectivity into existing devices.” [BTG01a, BTG01b, p. 1] “BlueGiga’s WRAP product family includes the WRAP 2100 Microservers and WRAP 3000 Industrial Access Server working together to enable three levels of machine interaction.” [BTG01c, p. 2]

“In the Man-to-Machine use case Bluetooth enabled PDAs (Personal Digital Assistant), mobile phones and laptops are used to wirelessly interface with machines. Traditional user interfaces (UI) such as LCD displays, monitors and keyboards are replaced with wirelessly distributed graphical interfaces such as HTTP with Java or Wireless Application Protocol (WAP).” [BTG01b, p. 2; BTG01c, p. 2]

“Man-to-Machine use case eliminates the need for expensive cabling between equipment in industrial environments. Besides cable replacement it also adds full TCP/IP functionality in each industrial device.” [BTG01b, p. 3]
“Machine-to-Networks use case provides embedded network connectivity for global remote access, control and system diagnostics as well as real-time data logging. It reduces costs by enabling field service automation, preventive maintenance and remote system configuration.” [BTG01b, p. 3]

7 THE TESTING OF BLUEGIGA'S PRODUCT PROTOTYPES

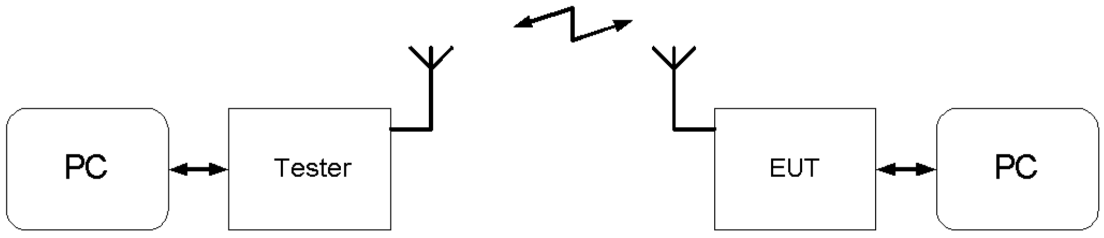
All the tests described in this chapter are black-box tests. The bit error test is done by comparing the input bits to the output bits and calculating the error rate. TTCN used in Man-to-Machine test case is a standardized language for describing black-box tests for reactive systems. The usage-based testing is purely functional and therefore black-box.

7.1 The Bit Error Rate Test According to IEEE Standards

At the beginning of the testing process a need for a constant way to perform and document the testing activities and results arose. The first solution was to document the software tests according to IEEE Standard for Software Test Documentation. This practice was evaluated during the bit error rate (BER) test.

Although, the bit error rate is not a Bluetooth test case listed in the test case reference list, the bit error rate under 0.1% is the expected outcome in many of the RF test purposes. BER measurements are carried out by comparing data in the payload fields transmitted by the tester with data in the payload fields received from the equipment under test (EUT). The EUT is in remote loop back mode [Blu01b, p. 812]. The BER test was done at BlueGiga facilities. The configuration of the test environment is presented in Figure 14. In this context, it is sufficient to say that the test was passed. A test binder was created to test the documentation process.

Figure 14. Test Configuration.



The BER test was documented according to the IEEE Standard for Software Test Documentation. The only item of documentation that was not created was the test-item transmittal report. This document was omitted, because the organization did not have a separate test facility to which to send the items for testing. The issues this document relates to, for example the status of the item, were taken care of with version control.

The creation of all the documents described in the IEEE Standard for Software Test Documentation was found to be extremely tedious. Therefore, it was decided that we would try to simplify the documentation process. This was to be done during the testing process, not beforehand, so that the new documentation process would be tested in practice simultaneously with its creation. This way, we would see straight-away if there was a risk of trying to take into use a too heavy documentation.

The first attempt to lighten the documentation was to create only the test design specification and the test summary report described in the IEEE Standard for Software Test Documentation. This is the minimum requirement for documentation stated in the IEEE Standard for Software Unit Testing.

The difficulty of applying the IEEE Standard for Software Test Documentation to the BER test might be due to the fact that it is based on the waterfall software development model. The internal software development model used at BlueGiga is the spiral model. The documents specified in the standard due not fit naturally into this framework. The waterfall model and standards based on it are not good as part of an internal testing (or the entire development) process.

The purpose was to practice for creating test documents that would be of use to customers and other external interest groups. Waterfall models strengths are in accountability and control that it provides to contract software. This is important as the customer projects begin at BlueGiga. The BER test was perhaps too much a truly internal test to practice on this type of process and documentation. It was anyhow useful as a learning experience for all participants.

7.2 Testing for Bluetooth Qualification

In this chapter, the issues related to the Bluetooth qualification are examined from BlueGiga's point of view. Important questions are: how do we choose a BQB (Bluetooth Qualification Body) and what needs to be qualified.

7.2.1 BQB and Test Facility Selection

Bluetooth Qualification Body (BQB) and Bluetooth Qualification Test Facility (BQTF) play an important part in the Bluetooth Qualification process. In BlueGiga's case these are both external facilities. Therefore, criteria for choosing them had to be developed. This was done via examining the information available on the Bluetooth qualification process and the qualification companies.

First of the criterion is the scope of testing performed by the test facility in terms of requirements for Bluetooth qualification. The requirements are split into four categories: Bluetooth radio link, protocol, profile, and information requirements. The most significant categories for BlueGiga are the radio link and profile requirements.

The testing of the radio link requires that the test facility has the necessary equipment and competence for RF testing. It would be practical if the test facility was certified to test also for government regulatory type approvals needed for the Bluetooth radio. Bluetooth radios operating in the ISM (Industrial Scientific Medical) band for Europe and America will need testing to ETS 300-328, ETS 300-826, and FCC§15 [Bra01, p. 395].

Profile testing is done against other devices supporting the profile(s). This way, the interoperability of the Bluetooth devices is tested. In profile testing, Blue Units, reference implementations of Bluetooth, are needed. The difficult availability, and high price, of Blue Units is a problem also in the internal testing, because without Blue Units testing can only be done against own or someone else's implementations of the Bluetooth specification, e.g. there is no reference implementation. Therefore, there is no guarantee that the implementation used in testing is conformant to the Bluetooth specification. Actually, even testing against a Blue Unit does not prove conformance. But, because of the availability and price problems of Blue Units, the Blue Unit testing is usually, and also in BlueGiga's case, performed by a Bluetooth Qualification Test Facility (BQTF).

Lesser test facility selection criterion are the test facility's capability to provide training in Bluetooth qualification and testing and availability of support for R&D –testing.

Even if the BQB by definition is an individual, he/she is usually affiliated with a test facility, or a company. Thus, the selection of a BQB is tightly connected with the selection of a test facility.

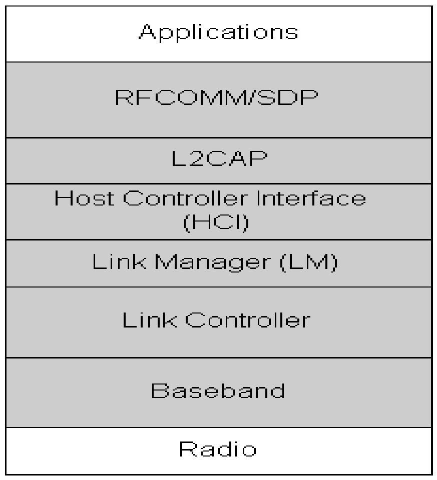
7.2.2 What Needs to be Qualified ?

In BlueGiga's End Product, i.e. WRAP product family, both a proprietary Bluetooth Component and listed Pre-Tested Bluetooth Components are used. The BlueGiga proprietary Bluetooth Component is the Bluetooth radio.

The BlueGiga WRAP product family implements one supplementary Bluetooth profile, in addition to those already qualified by the Bluetooth software stack manufacturer. This is the LAN Access Profile.

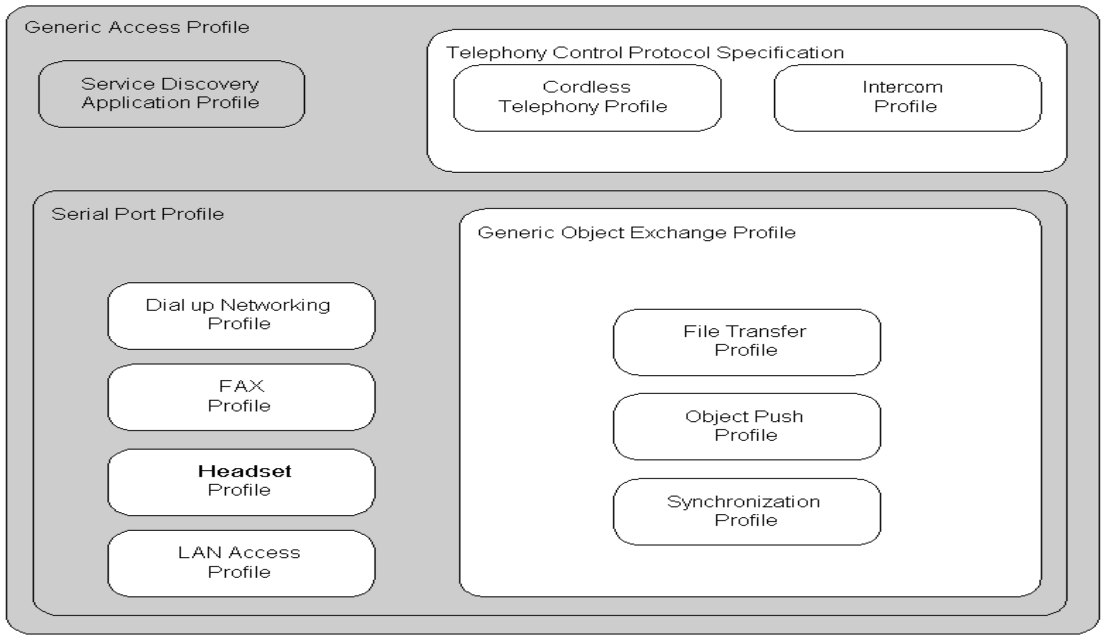
In Figure 15, you can see which parts of the Bluetooth stack have been Pre-Tested in the BlueGiga WRAP product family. Applications are out of the scope of the Bluetooth Specifications and thus, need not to be qualified.

Figure 15. Pre-Tested Components.



In Figure 16, you can see which profiles have already been qualified. These are the Generic Access, Serial Port and Service Discovery Application Profiles. The profile that is implemented in the WRAP product family in addition to these, is the LAN Access Profile and that has to also be qualified.

Figure 16. Qualified Profiles.



An ICS (Implementation Conformance Statement) should be filled both for the Bluetooth radio and LAN Access Profile. Based on the ICSs applicable test cases are selected from the corresponding Test Specifications with the help of the test case mapping table found at the end of each Test Specification. The test category (A, B, C or D) of a test case can be seen from the Test Case Reference List (TCRL).

The selected BQB can help with filling in the ICSs, selecting applicable test cases and identifying the test categories. Category A test cases have to be performed at a qualified Bluetooth Qualification Test Facility (BQTF). All other test cases may be performed either at the BQTF or at the manufacturer's own test facilities.

As long as the Pre-Tested Components are not affected by the adding of the Bluetooth radio and the implementation of the LAN Access Profile, they do not need to be re-tested. The evaluation of the need for re-testing is done by the BQB. In any case, it is not likely to be as wide as in the case of qualifying non Pre-Tested Components. The same applies for profiles.

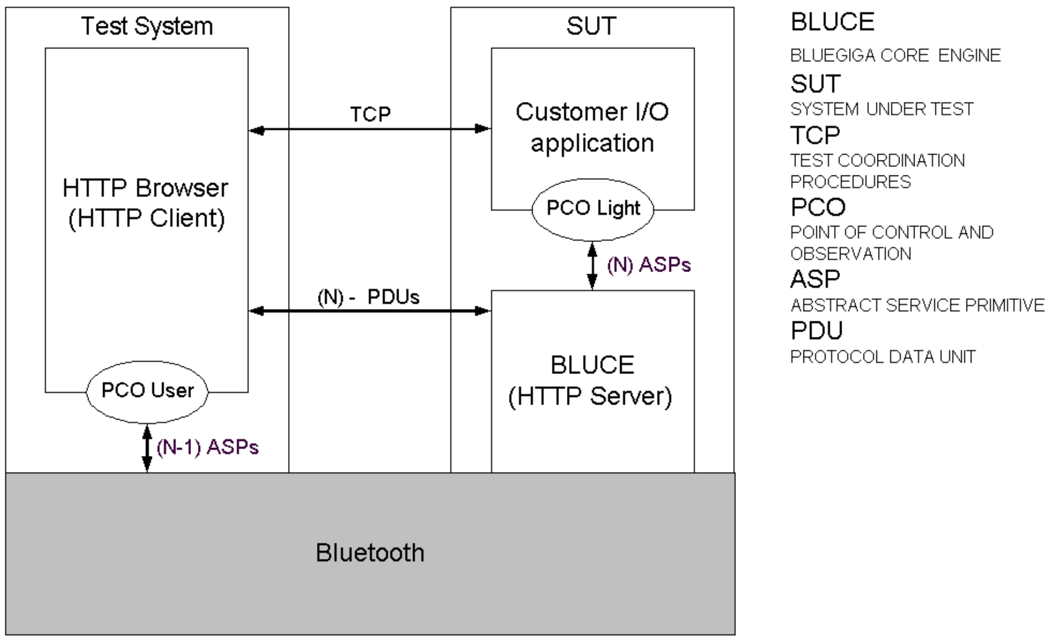
7.3 TTCN Test Case: Man-to-Machine

There were mainly two reasons for using TTCN in defining this test case. The first was the desire to get rid of ample written test documents. The second was the discovery that companies involved in the Bluetooth qualification process used TTCN based testing tools.

Man-to-Machine test case was defined with TTCN. It should be noted that the idea was to create an abstract human readable test case, not to develop an executable test suite. Therefore, only those parts of TTCN that were considered necessary to create an understandable and readable test case were used. These parts are listed in Appendix 2.

The TTCN definition of the abstract test architecture was found to be reasonably understandable when presented as a figure instead of a multitude of tables. The Man-to-Machine conformance test architecture is presented in Figure 17.

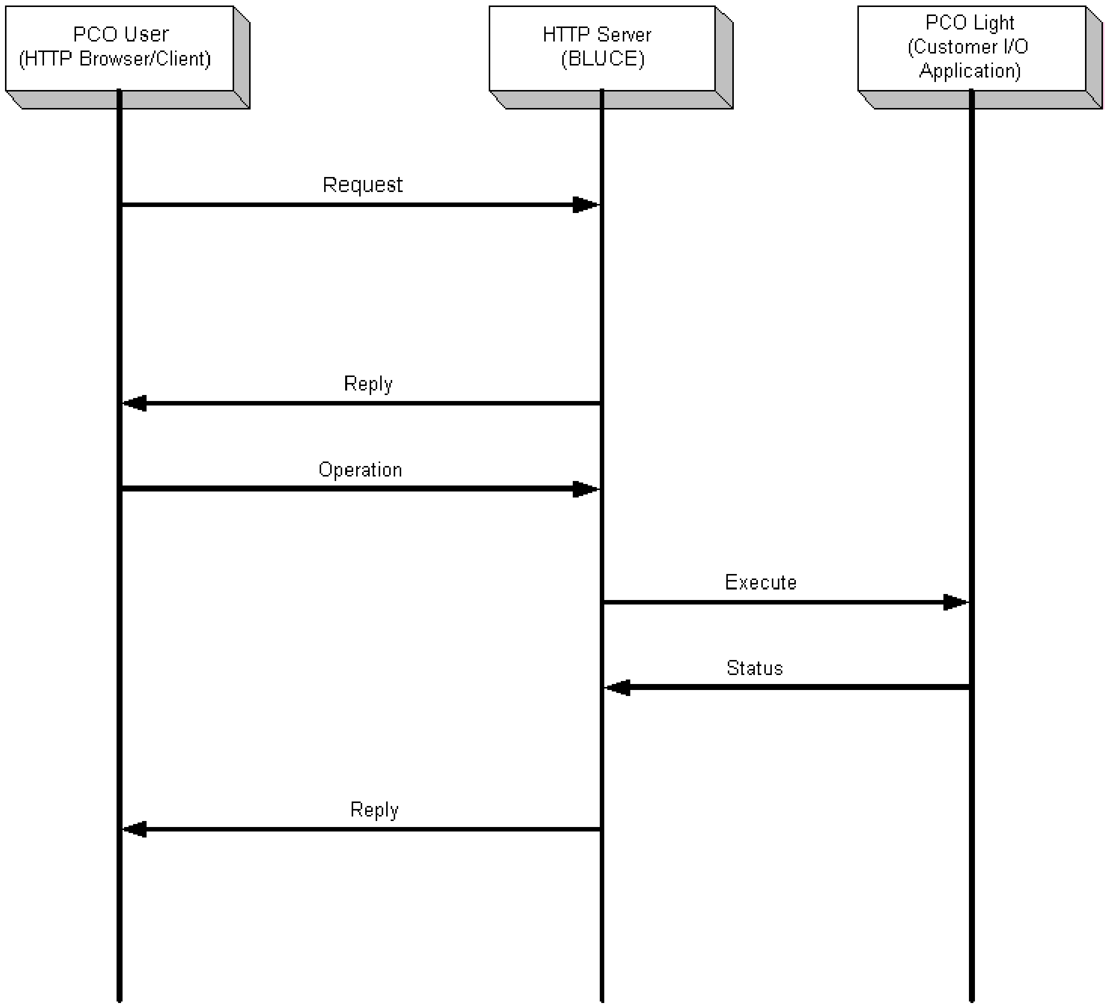
Figure 17. Man-to-Machine Conformance Test Architecture.



As you can see from Figure 17, the BlueGiga Core Engine BLUCE is the implementation under test (IUT). Customer I/O application is the upper tester (UT). HTTP client is the lower tester (LT). HTTP client and server are the peer protocol entities which exchange protocol data units (PDUs). Bluetooth is the (N-1) – service provider.

A MSC was drawn to aid in the developing of the actual TTCN Test Suite. The MSC, in Figure 18, was found to be extremely useful in explaining the test case to people familiar with the use case, but not knowing TTCN concepts and syntax. The MSC also fastened and smoothened the process of explaining the test case to people familiar with TTCN, but not knowing the use case.

Figure 18. The MSC of Man-to-Machine Test Case.



Three different Test Case Dynamic Behaviors were defined in the dynamic part. These were the Status Query, Operation Lights ON and Operation Lights OFF. Their purposes are presented in Table 6.

Table 6. The Purposes of the Test Cases.

Test Case	**Purpose**
Status Query	To test that the user gets the requested information, i.e. the status of the lights
Operation Lights ON	To test that the lights ON operation requested by the user is performed on the lights and that the user gets the correct status of the lights after the lights ON operation has been performed
Operation Lights OFF	To test that the lights OFF operation requested by the user is performed on the lights and that the user gets the correct status of the lights after the lights OFF operation has been performed

The Test Case Dynamic Behavior table for Operation Lights ON is presented in Figure 19. To walk through the Test Case Dynamic Behavior you should first look at the message sequence chart in Figure 18. There you can see the protocol data units (PDUs) relevant to this test case's dynamic behavior, Operation and Reply. The abstract service primitives (ASPs) relevant to this test case are Execute and Status, they are also presented in Figure 18.

The Operation is sent from the HTTP Client to the HTTP server, which is part of the BlueGiga Core Engine, BLUCE. The Execute is sent from the HTTP server to the Customer I/O application. The Status is sent from Customer I/O application to the HTTP server. The Reply is sent from HTTP server to HTTP client. Now we must remember that the PCO User is a part of lower tester, which is the HTTP client. The PCO Light is part of the upper tester, which is the Customer I/O application. The PCOs can be seen in Figure 17.

In the next paragraphs, we are going to use the terms PCO User and PCO Light. PCO User refers to the PCO located in HTTP client, PCO Light refers to the PCO located in Customer I/O application.

The Operation is sent from the PCO User to the HTTP server. At the PCO Light, it is checked if the Operation is OnOperation. If it is the preliminary verdict is pass, which is then marked as (P) to the verdict column. The other alternatives are that the operation is OffOperation, in which case the verdict is F, fail, which is a final verdict. If the Operation is something else the verdict is automatically F.

If the preliminary verdict was (P) in the previous phase the test case execution is continued from the next indentation level. Execute is sent from PCO Light to the Customer I/O application. The Customer I/O application performs the operation given as a parameter in Execute and sends Status to the PCO Light. If the Status is OnStatus the verdict is (P). If the Status is OffStatus the verdict is F, fail. If the Status is NoCurrentStatus, the verdict is (P). NoCurrentStatus means that the light is switched on, but there is no current. So, from the software point of view, everything is working properly. If the Status is FailureReply, the verdict is F. One could also see FailureReply as a valid event. Then there would have to be defined some reason for this error and this should be reported in the FailureReply. In that case the verdict for receiving FailureReply would be (P). If anything else is received the status is F.

If the previous phase of the test case execution was passed, Reply is sent from PCO Light to PCO User. If the Reply is OnReply or NoCurrentReply the final verdict is P, pass. If the Reply is OffReply or FailureReply the final verdict is F, fail. To the FailureReply applies the same as to FailureStatus in previous phase. If anything else is received the verdict is F. In that case, a protocol error has occurred, and the system has failed the test case.

Figure 19. Test Case Dynamic Behavior Example: Operation Lights ON.

Test Case Dynamic Behavior					
Test Case Name: Operation Lights ON					
Group:					
Purpose: To test that the lights ON operation requested by the user is performed on the lights and that the user gets the correct status of the lights after the lights ON operation has been performed.					
Configuration: Man-to-Machine					
Defaults:					
Comments: L=Label, Cref=Constraints Ref, V=Verdict, C=Comments					
Nr	**L**	**Behavior Description**	**Cref**	**V**	**C**
1		PCO User!Operation	OnOperation		
2		PCO Light?Operation	OnOperation		
3		PCO Light?Operation	OffOperation		
4		PCO Light?OTHERWISE			
5		PCO Light!Execute			
6		PCO Light?Status	OnStatus		
7		PCO Light?Status	OffStatus		
8		PCO Light?Status	NoCurrentStatus		
9		PCO Light?Status	FailureStatus		
10		PCO Light?OTHERWISE			
11		PCO Light!Reply			
12		PCO User?Reply	OnReply		
13		PCO User?Reply	OffReply		
14		PCO User?Reply	NoCurrent		
15		PCO User?Reply	FailureReply		
16		PCO User?OTHERWISE			
Detailed Comments:					

Already at very early phases of the writing of this test case using TTCN it was realized that the result would not be a more readable test case than a text-based one, but rather the opposite. Despite of this, the overall protocol testing process which is also an important part of the Bluetooth qualification was somewhat clarified by this example test case. To get real benefits from using TTCN we should have used a testing tool to create an executable TTCN test suite.

7.4 Usage-Based Testing: Man-to-Machine

The usage-based testing suggested by Regnell [Reg99] is tried out here. The Man-to-Machine use case is extended into an operational profile using model transformation. The actual Man-to-Machine use case demonstration had evolved from the time of writing the use case into containing also other functions than just setting the light on and off. For the sake of keeping the example simple (and comparable to the TTCN approach) these modifications are left out.

7.4.1 Use Case

First, the use case written earlier by Pesola [Pes01] is modified to suite the presentation used by Regnell in 'Derivation of an Integrated Operational Profile and Use Case Model'. The major activities in creating a use case model are "summarized below:

- Elicit actors and their goals.
- Define use cases based on actors and their goals.
- Elicit scenarios for each use case.
- Describe the events in the scenarios." [Reg99, p. 122]

In the Man-to-Machine system, three actors can be identified, *guest*, *user* and *administrator*. These actors and their goals are shown in Table 7.

Table 7. Actors and Goals.

Actors	**Goals**
Guest	GG1 To monitor lights
User	GU1 To set lights on
	GU2 To set lights off
Admin	GA1 To set light parameters

Of course, also the user might want to monitor the lights before setting them on or off. The administrator might want to monitor the lights or set them on or off to aid in the setting of the parameters.

Table 8 includes the services of the Man-to-Machine example system.

Table 8. Services.

Service	**Description**
ML	Monitor lights
SLON	Set lights on
SLOFF	Set lights off

SP	Set parameters

Table 9 shows use cases of the Man-to-Machine example, and their relation to actors, goals, and services.

Table 9. Use Cases.

Use cases	**Actors**	**Goals**	**Services**
Monitor lights	Guest	GG1	ML
Set lights on	User	GG1, GU1	ML, SLON
Set lights off	User	GG1, GU2	ML, SLOFF
Set light parameters	Admin	GG1, GU1, GU2, GA1	ML, SLON, SLOFF, SP

Table 10 shows a number of scenarios identified for the use case set lights on.

Table 10. Scenarios for Use Case ‘Set lights on’.

Scenario	**Description**
Success	The lights are set on successfully.
No Current	The lights are set on, but there is no current (a failure not related to the system).

The messages for the Man-to-Machine example are shown in Table 11 and the system actions can be seen in the example scenario described in Table 12.

Table 11. Messages.

Message	**Description**
statusQuery	From Guest, User or Admin when monitoring the lights.
statusReply	To Guest, User or Admin when the status of the lights is returned.
lightsOn	From User or Admin when setting the lights on.
lightsOff	From User or Admin when setting the lights off.
setParameters	From Admin when setting the parameters.

Table 12. Scenario Success of Use Case ‘Set lights on’.

Actor	User	
Pre-condition(s): Bluetooth link is not up.		
	Events	Constraints
1.	User to System: statusQuery	
2.	System action: establish Bluetooth link	
3.	System to User: statusReply	
4.	User to System: lightsOn	
5.	System action: Set lights on	
6.	System to User: statusReply	
7.	System action: close Bluetooth link	
Post-condition(s): Bluetooth link is closed.		

7.4.2 Derivation of an Operational Profile using Transformation Approach

First the **customer type** list is derived. Since there are no high-level actors in the use case model, there is no information on customer types, so this information has to be collected elsewhere. Generally the different customer types are defined as a requirement written in plain text. In our example, we have not yet defined customer types.

The **user type** list can be derived directly from the list of **actors** in Table 7. The actors *guest*, *user*, and *admin* are selected as user types.
Next, the **system modes** are elaborated. The information can be taken partly from the **services** in Table 8: *monitor lights* (ML), *set lights on* (SLON), *set lights off* (SLOFF) and *set parameters* (SP). In addition to this, we add the usage frequency for each service. The usage frequency of SLON and SLOFF is the same, because for every time the lights are switched on they are also switched off. If we assume that the user checks the status of the lights every time before switching them on or off, the usage frequency of ML is twice the combined usage frequency of SLON and SLOFF. It is reasonable to assume that the usage frequency of SP is considerably lower than any other usage frequency. We could also add here statistical information about the usage frequencies of each service according to customer types if it was available.

The list of **functions** is derived directly from the list of **use cases**, see Table 9. There are in total four functions, *monitor lights*, *set lights on*, *set lights off* and *set parameters*.

The **key input variables** which define variants of the functions can be derived from the **scenarios** of the use case, see Table 10. For the set lights on function, the key input variable is the success, partial success (no current) or failure of the operation. The levels of the variable are *success*, *no current* and *failure*.

"The usage frequency information in the **functional profile** cannot be found in the use case model, but has to be derived from other sources, for example interviews or measurements on existing systems [Reg99, p. 133]."

The functions are mapped onto **operations** which together constitute the function to the user. The **system actions** in the use case model constitute candidates for operations, see Table 13. In the set lights on use case, scenario, success, there are three system actions involved, *establish Bluetooth link*, *set lights on* and *close Bluetooth link*.

Finally, the functional profile is mapped onto the operations, defining the **operational profile**.

Table 13. Functions and their Mapping onto Operations.

Monitor light	Establish Bluetooth link Monitor lights Close Bluetooth link
Set lights on	Establish Bluetooth link Set lights on Close Bluetooth link
Set lights off	Establish Bluetooth link Set lights off Close Bluetooth link
Set parameters	Establish Bluetooth link Set parameters Close Bluetooth link

7.4.3 Derivation of an Operational Profile using Extension Approach

"On the **environment level**, we have to extend the use case model with information on the number of instances of each actor and the probabilities for each actor to generate stimuli to the system. Furthermore, it has to be analyzed if there are variants of actors with respect to their usage profile." [Reg99, p. 111] The information in Table 14 is purely an estimate. Actual user data for the example system will have to be collected so that this can be made realistic.

Table 14. Added Information to the Use Case Model on the Environment Level.

	Actor		**Variants**		**Use cases**
<0.05>	Guest (1)		none	<1.00>	Monitor lights
<0.90>	User (3)		none	<0.50>	Set lights on
				<0.50>	Set lights off
<0.05>	Admin (1)		none	<1.00>	Set parameters

"On the **structure level** we have to add profile information to the scenarios [Reg99, p. 112]." Again, the probabilities in Table 15 are only estimates. User information will have to be collected to get the actual probabilities for each scenario.

Table 15. The Use Case 'Set light on' Extended with Profile Information.

Scenario	Probability	Description

Success	<0.95>	The lights are set on successfully.
No Current	<0.05>	The lights are set on, but there is no current (a failure not related to the system).

On the **event level**, probabilities are attached to each choice in the model. The scenarios are then used as test cases. [Reg99, p. 112] The events are of three types: "**stimuli** (messages from user to the target system), **responses** (messages from the target system to users), and **actions** (target system intrinsic events which are atomic in the sense that there is no communication between the target system and the users that participate in the use case)". [Reg99, p. 96]

Success is a scenario where the Bluetooth link has been established, the data is transmitted, the light bulb is working, the answer returned to the user about the status of the lights is correct, and the Bluetooth link has been closed properly. If any of the before mentioned events fails, the system has failed the test. *No current* is a special case where the data to set the lights on has been transmitted properly, but the light bulb is broken and the system informs the user of the broken lamp.

7.4.4 Comparison of the Approaches

The transformation approach has the advantage of being based on two relatively mature disciplines use case modeling and operational profiles. The major disadvantage of this approach is that it leads to two models which have to be maintained. There is a risk of inconsistencies between the models. On the other hand the two models are both tailored to fit their purposes. As a consequence, the models are at appropriate level of detail. [Reg99, p. 137]

In the extension approach, there is only one model that needs to be derived and maintained [Reg99, p. 136]. This approach would be more suitable to BlueGiga. The model is, anyhow, a compromise between purposes and, therefore, a compromise between the levels of detail. This problem can be tackled by setting the level of details for the requirements purposes in the beginning of the development and evolving it into more details, according to the test model needs. [Reg99, p. 136-137]

8 THE BLUEGIGA TECHNOLOGIES' TESTING PROCESS

BlueGiga Technologies' testing process is based on the Boehm's spiral software development model. At the end of each development cycle, testing takes place. Following the principles of extreme programming, tests should be created before the code is written, while the code is written and after the code is written. The goal is to create unit tests before the code is written. The integration and system tests are created while the code is written. This leads us to the question what tests are there left to be created after the code is written? These are the tests that are performed only at the end of the next development cycle. These tests take advantage of the code written in order to develop the current prototype.

8.1 Unit and Component Testing

The unit testing is done by the programmers. Only working and tested versions of the software units should be entered to the version control. Attention should be paid also to the comments both in version control entries and in the code itself. The software units entered to the version control were mostly working or there was enough grounds to enter them to the version control for some other reason, so this was not really a problem.

The level of the documentation in the code and in the version control entries is highly dependent on the individual programmer. There is always room for improvement in the commenting habits of individuals. It is almost impossible to impose these habits on individuals from the outside, so there is not much a testing process can do to improve this.

The issue of documenting the code and version control entries is closely related to the documentation of tests performed on the software unit. As a solution to the documentation of the tests, it was suggested that the extreme programming practice of writing the test programs even before writing the programs themselves should be taken into use. The decision to use this practice or not to was left to the individual programmers. If the test programs indeed are written, they should be checked into the version control together with the program under test.

8.2 Integration Testing

The testing of hardware components before integrating them with the software is out of the scope of this process description. The integration testing of the software units all coded by the same programmer is his responsibility. The working of the interfaces between software units coded by different programmers is also the responsibility of the corresponding programmers.

At the point, where hardware and software are integrated, the kernel is loaded to the flash. After this, it is checked that the flash is working properly. The kernel with Wireless Remote Access Platform (WRAP) software is installed. After this, it is tested that the kernel boots as it should.

This really fills the definition of integration testing by Beizer [Bei84, Bei90], because the successfulness of the integration itself is tested and not the working of the integrated components. The testing of the integrated system as a whole is correctly left to the system testing phase. Also, the responsibility for the integration testing is left to the programmers, as Beizer [Bei84] suggests.

8.3 System Testing

After successful software to hardware integration, the board under test is given both MAC (Medium Access Control) and Bluetooth addresses. These are not the final addresses, but addresses reserved for testing purposes.

The self-test software is loaded from the CD. Then the self-test is run. The self-test shows that all the LEDs on the board are working. After successful self-test the board can be pinged from a PC. A Bluetooth connection can be made from another device and the connection can be tested by running the BER test or doing a simple file transfer.

The usage-based part of the system testing is the use of Man-to-Machine prototype hardware and software in testing the functionality of the entire integrated system. The prototype hardware board is connected to the board under test and software is loaded from the CD. After this, ethernet and Bluetooth connections are made from the board under test to the demo hardware board. Now, all the features available in the prototype can be exercised. These include the monitoring and controlling of a lamp described in the Man-to- Machine use case.

System testing is done by independent testers. Independent testers can be used efficiently at this phase, because no information about the internal structure of the system is needed. It is also more probable that different features of the software will be exercised by independent testers than if the testing was done by the programmers. Programmers would be prone to exercise the same features as in the unit testing phase.

At the system testing phase, usability testing could be done to see if the loading of the software from the CD is user-friendly and straight-forward. Also, the software examples delivered together with the CD need to be tested for usability.

During the development process an idea of writing a test bench was born. The test bench could be used for testing the BlueGiga Technologies middleware as an independent software component, not integrated with the Bluetooth protocol stack. This way it would be easier to locate errors.

9 CONCLUSION

It is quite commonly agreed that the testing process should be divided into the following three phases: unit and component testing, integration testing and system testing. The optimal distribution of work among the programmers and independent testing personnel is not that clear. We ended up in using programmers in unit, component and integration testing and independent testers in system testing. This approach is suggested, for example, by Beizer [Bei84, Bei90] and Wells [Wel01].

The programmers knowledge of the units and components makes it possible to create tests faster than if independent testers were used. The programmers also have the best knowledge of the interfaces which is needed in rapid integration testing. System testing has a user's point of view to the testing, so it can be efficiently done by someone who does not know the inner workings of the system. It is psychologically easier for an outsider to spot the weak spots of the system and stress them than it is for the programmers.

Testing is part of the entire software development process. Therefore, it must be performed in a way that does not conflict the larger process. The application of the waterfall model into testing was difficult when the overall software development was based on the spiral model. The spiral model is suitable for intensive product development, because it supports rapid prototyping. The waterfall model has, anyhow, even at this day and age, its benefits in providing accountability and control to contract software.

When the product development is intense, heavy process and documentation required by older models, as, for example, the waterfall model, slows down the work. Light-weight software development processes, like Extreme Programming (XP), have been developed to tackle this problem. They are a lot more suitable for small new companies than old, heavy software development processes.

Tree and Tabular Combined Notation (TTCN) was studied and found to be useful in providing a framework for defining tests. It was, however, not taken into use due to lack of resources. The main benefit was the increased understanding of testing as it is done in the Bluetooth Qualification Test Facilities (BQTFs).

A new technology is always a challenge to testing. In Bluetooth technology in particular, there are interoperability issues that need to be solved. The heavy qualification process is a burden to the Bluetooth technology, especially from the point of view of small companies that do not have they own test facilities that could be qualified for Bluetooth testing. The qualification process does not even guarantee interoperability, but only conformance to the specification. Conformance to the specification should, however, increase the changes of interoperability between devices from different manufacturers.

Usage-based testing was found to be very useful at system level conformance to requirements testing. When use cases for the system are defined at the requirements engineering phase, it is practical to use them later as a basis for designing test cases. This requires that the use cases are kept up to date as the system evolves. The level of detail in the use cases should also increase as the system elaborates.

The usage-based testing suggested by Regnell has a lot of similarities to the user stories used as the base for acceptance testing in Extreme Programming (XP). It would be interesting to carefully compare these approaches in the future.

REFERENCES

[IEEE86] ANSI/IEEE 1008-1987. 1986. IEEE Standard for Software Unit Testing. New York. IEEE. 24 pp.

[IEEE83] ANSI/IEEE 829-1983. 1983. IEEE Standard for Software Test Documentation. New York. IEEE. 48 pp.

[Bei84] Beizer, Boris. 1984. System Testing and Quality Assurance. New York, USA. Van Nostrand Reinhold. 358 pp. ISBN 0-442-21306-9.

[Bei90] Beizer, Boris. 1990. Software testing techniques. USA. International Thomson Computer Press. 550 pp. ISBN 1850328803.

[BGT01a] BlueGiga Technologies, The Official Website, [Referenced: October 11, 2001] n. pag. Available: http://www.bluegiga.com

[BGT01b] BlueGiga Technologies. 2001. Remote Access Solutions for the Industry. WRAP Product Brochure. 4 pp. Available: http://www.bluegiga.com

[BGT01c] BlueGiga Technologies & Atmel. 2001. Atmel and BlueGiga Technologies Bring Wireless Connectivity Solution to OEMs. News Release, September 26, 2001. San Jose & Espoo. 3 pp.

[Bha00] Bhagwat, Amit. 2000. Architecture and Design: Object-Oriented Analysis and Design Methods [www-document]. Cetus Team. n. pag. [Last modified: September 30, 2000] [Referenced: December 7, 2001] Available: http://www.sente.ch/cetus/oo_ooa_ood_methods.html

[Blu01a] Bluetooth Special Interest Group, The Official Website n. pag. [Referenced: September 28, 2001] Available: http://www.bluetooth.com

[Blu01b] Bluetooth Special Interest Group, "Bluetooth Core", Specification of the Bluetooth System, Version 1.1, February 22, 2001. 1084 pp. Available: http://www.bluetooth.com/developers/specification/Bluetooth_11_Specification_Book.pdf

[Blu01c] Bluetooth Special Interest Group, "Bluetooth Profile", Specification of the Bluetooth System, Version 1.1, February 22, 2001. 452 pp. Available: http://www.bluetooth.com/developers/specification/Bluetooth_11_Profiles_Book.pdf

[Blu01d] Bluetooth Special Interest Group, "Qualification Program Reference Document", Revision 0.9, August 10, 2000, 75 pp. Available for Bluetooth SIG members: http://www.bluetooth.org/member/docs/ Qualification_Program_Reference_Document_21Aug00.pdf

[Bra01] Bray, Jennifer & Sturman, Charles F. 2001. Bluetooth Connect without Cables. Upper Saddle River, USA. Prentice-Hall. 495 pp. ISBN 0-13-089840-6.

[Cet01] Cetecom. 2001. All About Bluetooth Qualification, Volume 3, Part 9: Tree and Tabular Combined Notation (TTCN) [Lecture Slides]. 98 pp.

[Hak94] Hakulinen, K. 1994. Solukkoverkkoprotokollien tietokoneavusteinen testaus. Master's Thesis. Helsinki University of Technology. 79 pp.

[Hel01] Helsinki University Library. 2001. Linda Database – Union Online Catalogue of the Academic Libraries. Online. n. pag. [Referenced: November 23, 2001] Available: http://linneaw.lib.helsinki.fi/suora_linnea/

[ISO98] ISO/IEC 9646-3. 1998. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). Switzerland. ISO/IEC. 265 pp.

[Juk99] Jukkara, Pasi. 1999. MSC testing over multiple interfaces using Modular and Concurrent TTCN. Master's Thesis. Lappeenranta University of Technology. 52 pp.

[Kan97] Kan, Stephen H. 1997. Metrics and Models in Software Quality Engineering. Fourth edition. USA. Addison-Wesley Longman, Inc. 344 pp.

[Mil01] Miller, Brent A. & Chatschik, Bisdikian. 2001. Bluetooth Revealed. Upper Saddle River, USA. Prentice-Hall. 303 pp. ISBN 0-13-090294-2.

[Pes01] Pesola, Juuso. 2001. BlueGiga Use Cases: Man-to-Machine, Machine-to-Machine, Machine-to-Networks. Espoo, VTT. 16 pp.

[Pre00] Pressman, Roger S. 2000. Software Engineering, A practitioner's Approach, European adaptation. Adapted by Darrel Ince. Fifth edition. Cornwall, UK. McGraw-Hill International (UK) Limited. 915 pp. ISBN 0 07 709677 0.

[Reg99] Regnell, Björn. 1999. Requirements Engineering with Use Cases – a Basis for Software Development. Doctor's Thesis. Lund University. 225 pp.

[Rob01] Roberts, Mike, Editor. 2001. Qualification does not equal interoperability. Planet Wireless, 2001. pp. 13-15.

[Sai00] Sainio, Liisa-Maija, Sikiö, Taina. & Niiranen, Jukka. 2000. Application Visions and Business Opportunities of Bluetooth – A Wireless Technology for Local Data Transfer. Lappeenranta: Telecom Business Research Center. 35 pp. (Telecom Business Research Center, Working Papers 5.) ISBN 951- 764-482-5.

[Sch98] Schneider, Geri. Winters, Jason P. 1998. Applying Use Cases, A Practical Guide. USA. Addison-Wesley Longman, Inc. 188 pp. ISBN 0-201-30981-5.

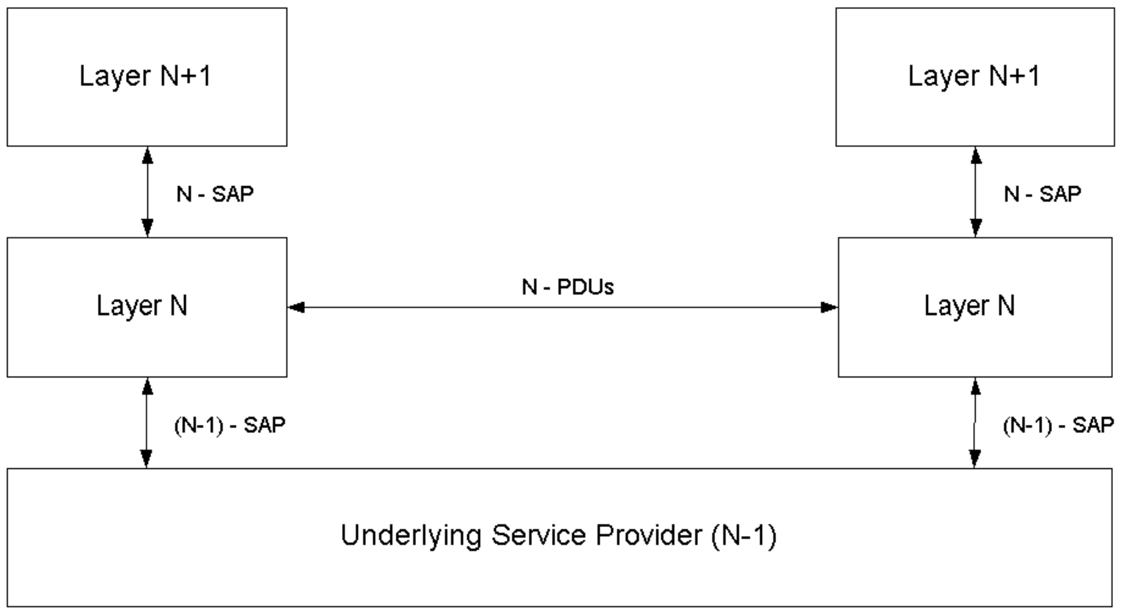
[TL99] TL 9000. 1999. TL 9000 Quality System Requirements, Book One, Release 2.5. Quality Excellence for Suppliers of Telecommunications Leadership (QuEST) Forum. 49 pp.

[Toi01] Toivonen, Aki. 2001. Bluetooth –järjestelmän laboratoriotestauksen vaatimusten määrittely. Special work. Turku Polytechnic. 44 pp.

[Wel01] Wells, J. Donovan. 2001. Extreme Programming: A gentle introduction [www-document]. n. pag. [Last modified: June 4, 2001] [Referenced: September 14, 2001]. Available: http://www.extremeprogramming.org

[Wil00] Wiles, Anthony. 2000. TTCN Factsheet [www-document]. ETSI: Protocol and Testing Competence Centre. n. pag. [Last review: March 1, 2000] [Referenced: July 31, 2001]. Available: http://www.etsi.org/ptcc/ptcc_ttcn.htm

Appendix 1. The OSI Model.



Appendix 2. The Parts of TTCN Used in Man-to-Machine Test Case.

The Man-to-Machine test suite does not import anything, so the Import Part is not needed. This can be seen from Table 1.

Table 1. Parts of a TTCN test suite.

Suite Overview	●
Import Part	
Declarations Part	●
Constraints Part	●
Dynamic Part	●

As you can see from Table 2 below, only Test Suite Structure was used. Test Case Index was not used, because it was considered to be overkill to use a Test Case Index for only three test cases. Test Step and Test Default Indices were not used, because no test steps or defaults were defined. Test Suite Exports and Imports were not used, because there is no other test suite that would import from this test suite nor does this test suite import from another test suite.

Table 2. Test Suite Overview Part.

Test Suite Overview Part	
Introduction	
Test Suite Sructure	●
Test Case Index	
Test Step Index	
Default Index	
Test Suite Exports	
Imports	

(appendix 2 continues)

What you can see from Table 3 below is that a non-concurrent conformance test architecture and the PDUs and ASPs used in it plus their data types were declared.

Table 3. Declarations Part.

Declarations Part	
Definitions	
Test Suite Types	●
Test Suite Operations	
Parameterization and selection of Test Cases	
Test Suite Parameters	
Test Case Selection Expressions	
Declarations/definitions	
Test Suite Constants	
Test Suite Variables	
Test Case Variables	
PCO Types	●
PCOs	●
CPs	
Timers	
Test Components	●
Test Component Configurations	●
ASP Types	●
PDU Types	●
Encoding Rules	
Encoding Variations	
Invalid Field Encodings	
CM Types	
Aliases	

In the Constraints Part constraints for the PDUs and ASPs were defined.

(appendix 2 continues)

Test Cases were so small that there was no need to modularize them by using Test Steps. Defaults could have been useful, but to maintain readability they were not used. This is presented in Table 4.

Table 4. Dynamic Part.

Dynamic Part	
Test Case Dynamic Behaviour	●
Test Step Dynamic Behaviour	
Default Dynamic Behaviour	

Appendix 3. The Bluetooth Profiles.

The current Bluetooth profiles are listed in Table 1. Those applicable to the WRAP (Wireless Remote Access Platform) product family are marked with a dot.

Table 1. Bluetooth Profiles Applicable to the WRAP [Blu01c].

Generic Access Profile	●
Service Discovery Application Profile	●
Cordless Telephony Profile	
Intercom Profile	
Serial Port Profile	●
Headset Profile	
Dial-Up Networking Profile	
FAX Profile	
LAN Access Profile	●
Generic Object Exchange Profile	
Object Push Profile	
File Transfer Profile	
Synchronization Profile	