HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Electrical and Communications Engineering

Jorma Tuominen

Test process analysis of Gateway GPRS Support Node

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Espoo, 8th January 2006.

Supervisor	Prof. Sven-Gustav Häggman
Instructor	M.Sc. Rui Zhao

HELSINKI UNIVERSITY OF TECHNOLOGY	**ABSTRACT OF THE MASTER'S THESIS**

Author: Jorma Tuominen

Name of the Thesis: Test process analysis of Gateway GPRS Support Node

Date: 8.1.2006 **Number of pages:** 72

Department: Electrical and Communications Engineering

Professorship: S-72 Communications

Supervisor: Professor Sven-Gustav Häggman

Instructor: M.Sc. Rui Zhao

In this thesis the test process of Gateway GPRS Support Node was analysed by means of testing metrics, a defect analysis and a product risk analysis. Based on the before mentioned analyses a risk based testing was planned. The objective of the thesis was to improve the efficiency of the testing by concentrating the risk based testing to the features that are most critical to the customers and also by emphasising the testing of the most error prone functionality areas. The goal is a higher product quality and lower testing costs.

In the theoretical part of the thesis firstly the functionality of Gateway GPRS Support Node is explained and also the protocols and interfaces towards the other network elements are investigated. Secondly the basics of the software creation process and the software testing are described.

In the practical part of the thesis the testing metrics are researched and the defect analysis is performed. Then the risk based testing is planned based on the product risk analysis.

Main conclusions of the study: 1) Co-operation between the product development, the testing and the customer support should be strengthened. 2) The usage of the requirement management database, the test management database and the defect tracking database should be harmonized. 3) The availability of the real-time testing metrics should be improved.

Keywords: GGSN, software testing, testing metrics, defect analysis, risk based testing

TEKNILLINEN KORKEAKOULU	**DIPLOMITYÖN TIIVISTELMÄ**		
Tekijä:	Jorma Tuominen		
Työn nimi:	GPRS yhdyskäytäväsolmun testausprosessin analysointi		
Päivämäärä:	8.1.2006	**Sivumäärä:**	72
Osasto:	Sähkö- ja tietoliikennetekniikan osasto		
Professuuri:	S-72 Tietoliikennetekniikka		
Työn valvoja:	Professori Sven-Gustav Häggman		
Työn ohjaaja:	Diplomi-insinööri Rui Zhao		

Tässä diplomityössä tarkastellaan GPRS ja 3G verkon yhdyskäytäväsolmun testausprosessia käyttäen apuna erilaisia testauksen mittareita, vika-analyysiä ja riskianalyysiä. Edellä mainittujen analyysien perusteella suunnitellaan riskiperusteisen testauksen tarkka kohdentaminen.

Tavoitteena on löytää keinoja parantaa testauksen tehokkuutta kohdentamalla riskiperusteinen testaus asiakkaan kannalta tärkeimmille ja vika-altteimmille toiminnallisuusalueille. Tämän tuloksena asiakkaalle pystytään toimittamaan parempilaatuinen tuote ja testauksen kustannuksia pystytään alentamaan.

Työn teoriaosuudessa selvitetään ensin GPRS yhdyskäytäväsolmun toimintaa ja kommunikointia naapuriverkkoelementtien kanssa. Toiseksi selvitetään ohjelmistokehitysprosessin ja ohjelmistotestauksen perusteita.

Työn käytännön osuudessa tutkitaan testauksen mittareita ja tehdään vika-analyysi. Mittareihin ja vika-analyysiin perustuen tehdään tuotteen riskianalyysi, jonka jälkeen pystytään suunnittelemaan riskiperusteinen testaus.

Tutkimuksen tärkeimmät johtopäätökset: 1) Yhteistyötä tuotekehityksen, testauksen ja asiakasrajapinnan välillä pitää lisätä ja tehostaa. 2) Vaatimustenhallinnan, testitapaustietokannan ja vikatietokannan käyttöä pitää yhdenmukaistaa. 3) Testauksen ohjausta ja seurantaa helpottavien mittareiden saatavuutta ja reaaliaikaisuutta pitää kehittää.

Avainsanat: GGSN, ohjelmistotestaus, testauksen mittarit, vika-analyysi, riskiperusteinen testaus

PREFACE

This thesis was carried out at Network Service Core and Platforms, Nokia Networks, Helsinki.

I would like to thank Professor Sven-Gustav Häggman for supervising this thesis. His guidance and comments were a great support.

I would like to thank my instructor Rui Zhao and my superior Timo Vierimaa for the advices and the encouraging comments, department manager Mikko Ohvo for providing the opportunity to carry out this thesis at System Operations department. I would also like to thank all my colleagues in the highly professional compatibility and release testing team for their expertise.

Finally, I want to thank warmly my family and loved ones for their patience and support.

Helsinki, January 2006

Jorma Tuominen

TABLE OF CONTENTS

TERMS, ACRONYMS AND ABBREVIATIONS

2G	2nd Generation mobile communications
3G	3rd Generation mobile communications
3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorisation, Accounting protocol
AP	Access Point
APN	Access Point Name
BG	Border Gateway
BGP-4	Border Gateway Protocol 4
BS	Base Station
BSC	Base Station Controller
BSS	Base Station Subsystem
BTS	Base Transceiver Station
CA	Content Analyser
CDR	Charging Data Record
CG	Charging Gateway
CGF	Charging Gateway Function
CHAP	Challenge Handshake Authentication Protocol
CLI	Command Line Interface
CT	Compatibility Testing
CTRT	Compatibility and Release Testing
DHCP	Dynamic Host Configuration Protocol
Diameter	AAA protocol evolved from RADIUS
DNS	Domain Name Server
DRE	Defect Removal Effectiveness
EDGE	Enhanced Data rates for GSM Evolution
FT	Functional Testing
FTP	File Transfer Protocol
FW	Firewall
Ga	Interface between GPRS support node and charging gateway
Gb	Interface between BSS and Serving GPRS Support Node
G-CDR	GGSN Charging Data Record
GGSN	Gateway GPRS Support Node
Gi	TCP/IP based interface from GGSN to external data networks

Gn	Interface between GPRS Support Nodes within a PLMN
Gp	Interface between GPRS Support Nodes in different PLMNs
GPRS	General Packet Radio Service
GRE	Generic Routing Encapsulation
GSM	Global System for Mobile
GSN	GPRS Support Node
GTP	GPRS Tunnelling Protocol
GTP'	Enhanced GPRS Tunnelling Protocol
GTP-C	GPRS Tunnelling Protocol Control plane
GTP-U	GPRS Tunnelling Protocol User plane
HLR	Home Location Register
HPLMN	Home Public Land Mobile Network
HTTP	Hypertext Transfer Protocol
ICD	Intelligent Content Delivery
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPCP	IP Control Protocol
IPsec	Internet Protocol security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISD	ICD Service Directory
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
Iu	Interface between Serving GPRS Support Node and UTRAN
L2TP	Layer Two Tunnelling Protocol
L4	Layer Four of TCP/IP protocol stack
L7	Layer Seven of TCP/IP protocol stack
LAC	L2TP Access Concentrator
LCP	Link Control Protocol
LDAP	Lightweight Directory Access Protocol
LIG	Lawful Interception Gateway
LNS	L2TP Network Server

ME	Mobile Equipment 2G
MIB	Management Information Base
MMS	Multimedia Messaging Service
MS	Mobile Station 2G (ME+SIM)
MSISDN	Mobile Subscriber International ISDN Number
MT	Mobile Terminal
MT	Module Testing
NE3S	Nokia Enhanced SNMP Solution Suite
NetAct	Nokia Network Management System
NMS	Network Management System
NSM	Nokia Subscription Manager
NTP	Network Time Protocol
O&M	Operation and Maintenance
OSC	Online Service Controller
PAP	Password Authentication Protocol
PDN	Packet Data Network
PDP	Packet Data Protocol
PET	Performance Testing
PI	Product Integration
PLMN	Public Land Mobile Network
PoC	Push to talk over Cellular
PPP	Point-to-Point Protocol
P-TMSI	Packet Temporary Mobile Subscriber Identity
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RADIUS’	RADIUS prime, proprietary Nokia protocol used in OSC interface
RAN	Radio Access Network
RFC	Request For Comments
RNC	Radio Network Controller
RT	Release Testing
RTSP	Real-Time Streaming Protocol
RUGT	Release Upgrade Testing
SA-CDR	Service Aware CDR
SGSN	Serving GPRS Support Node

SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
ST	System Testing
SW	Software
SyVe	System Verification
TA	Traffic Analyser
TCP	Transmission Control Protocol
TE	Terminal Equipment
Telnet	Terminal emulation protocol
UDP	User Datagram Protocol
UE	User Equipment 3G
URI	Uniform Resource Identifier
UTRAN	Universal Terrestrial Radio Access Network
VPLMN	Visited Public Land Mobile Network
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background

Customer satisfaction is the primary goal of the product development process. The customer is satisfied if the price of the system is correct, time to market is correct and the system has features that the customer requires. Testing is significant part of the product development process and testing has effect on price, time to market and features of the system. About 40% of product development costs consist of testing costs, so the effect on price is obvious. In every development project the testing is on the critical path of the total development process, so the time to market depends on the time used for testing. The features of the system have been defined in the beginning of the project but testing is the phase, where it is verified that the promised functionality is really implemented and that the system satisfies the specified requirements.

In real-life systems the software complexity is so high that the exhaustive testing will take so long time that the market window will be closed before the testing is finished i.e. everything cannot be tested. Therefore it is crucial to analyse what, when, where, and how thoroughly the features has to be tested to find the optimal balance between the desired quality and the amount of time/money required. This kind of analysis is called software risk analysis. System features must be prioritised according the likelihood that the failure occurs, and according the impact to customer if the failure occurs in a particular area of the system.

The information needed when performing the software risk analysis must be gathered from several sources. The defects discovered by in-house testing of the previous product releases or in earlier levels of tests would show the error prone software modules. The defects reported by customers give valuable information about the features that are important to customers and are used by customers. All customer feedback and information about customer system configuration helps priorization of the features. Also the evolution of the product must be analysed. In how many previous releases each feature has existed and how each feature has been tested in earlier product releases.

1.2 Research problem

It is a fact that all defects cannot be discovered during the software testing. Also the software testing has always time and money constraints. Therefore, a significant problem is how available time and resources can be used as effectively as possible to find and correct the critical defects from the product before it is released?

1.3 Objectives of the study

The objective of the study is to collect necessary information to make a software risk analysis. By placing the emphasis of testing to the areas found by the software risk analysis the effectiveness of the testing will increase.

1.4 Scope of the study

The scope of this study is the system testing process of Nokia Gateway GPRS Support Node (GGSN). The system testing in this case can be divided into four areas: functional testing, compatibility testing, performance testing and release testing. The main focus is on compatibility and release testing. This study concentrates on the testing phases that are performed by independent testing teams. In case of the Nokia GGSN product creation process the programmers perform the module testing, the integration testing is performed by programmers together with functional testing personnel. Other testing phases are performed by independent testing personnel. The module testing is out of the scope of this study.

1.5 Research methods

This study was performed by using the following methods:

- Literature study of packet core networks and GGSN
- Literature study of software testing
- Analysis of the defects reported into the problem reporting system
- Analysis of the requirements of the GGSN releases
- Discussions with GGSN R&D, testing and customer care personnel
- Participating in the GGSN test planning, test design, test execution and test reporting

1.6 Structure of the thesis

Chapter 2 outlines the network architecture and the network elements of the mobile packet core network, and also the network elements of the Nokia Intelligent Content Delivery system are introduced.

More detailed introduction to the GPRS Gateway Support Node is given in Chapter 3. It is very important to understand how the GGSN communicates with the other network elements. Therefore the interfaces of the GGSN and used protocols are described.

Chapter 4 concentrates on the software testing as a part of the development process. This chapter gives some general information about the software testing and the guidelines of a risk based testing are introduced. Also the different testing phases are explained.

In Chapter 5 different kind of GGSN testing metrics are illustrated.

The defect analysis i.e. a method how to find the most error prone functionality areas of GGSN is explained in Chapter 6.

The product risk analysis is described in Chapter 7. The risk analysis is partially based on the information created in the defect analysis and partially based on a requirement analysis. For each requirement it is analysed what is the likelihood of failure and what the impact on the user is if the functionality defined in the requirement fails to operate.

Chapter 8 explains how the product risk analysis should be taken into account in test planning, test design and test execution phases.

Finally in Chapter 9 the results of the thesis are summarised and the recommendations for future actions are given.

2. GPRS AND 3G PACKET CORE NETWORK

This chapter outlines the network architecture of General Packet Radio Service (GPRS) and 3rd Generation (3G) packet core network. Also the network elements are introduced. In addition, the network elements of Nokia Intelligent Content Delivery system are presented.

2.1 Network architecture

The mobile packet core network is the connecting link between mobile radio networks and the packet based services in Internet or in Internet Service Provider (ISP) intranet or in corporate intranet. The mobile packet core network architecture is presented in Figure 1. The same mobile packet core network serves all three different mobile radio networks: GSM (Global System for Mobile), EDGE (Enhanced Data rates for GSM Evolution) and WCDMA (Wideband Code Division Multiple Access).

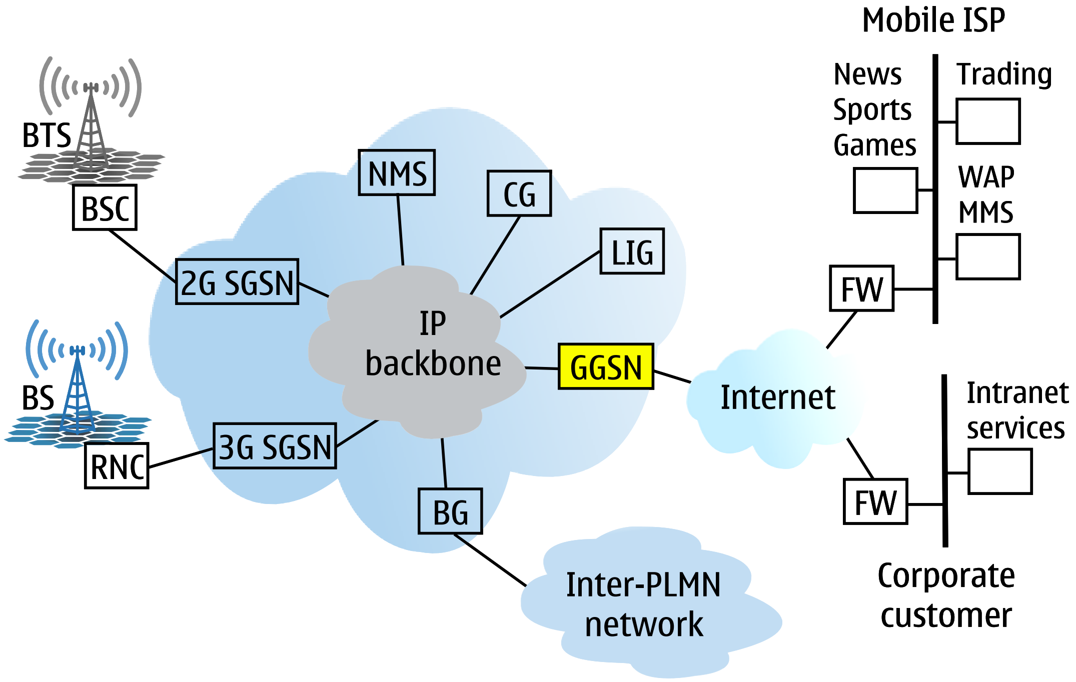

Figure 1 Mobile packet core network [1, p.33]

2.2 Network elements of mobile packet core

2.2.1 2G Serving GPRS Support Node

The 2G Serving GPRS Support Node (2G SGSN) connects the GSM and EDGE Base Station Subsystem (BSS) to the mobile packet core network.

2G SGSN takes care of subscriber mobility management (attach, detach, location management), session management and radio protocol to Internet Protocol (IP) conversion. It performs GTP (GPRS Tunnelling Protocol) tunnelling and routing and transfers the user data packets to the relevant GGSN. In addition 2G SGSN authenticates the subscriber and the Mobile Station (MS), and performs the ciphering and compression of user data. The 2G SGSN also generates charging data and transfers it to the Charging Gateway (CG) for billing purposes. As well 2G SGSN generates statistical information and alarm information to Network Management System (NMS). Upon request the 2G SGSN intercepts user data and location information for Lawful Interception. [2, p.13]

2.2.2 3G Serving GPRS Support Node

The 3G Serving GPRS Support Node (3G SGSN) connects the WCDMA Radio Access Network (RAN) to the mobile packet core network. The 3G SGSN performs mainly the same functions, as 2G SGSN except that 3G SGSN does not have to do the protocol conversion from radio protocol to IP because 3G SGSN uses the GTP protocol towards RNC (Radio Network Controller) and GGSN. [3, p.7]

2.2.3 Border Gateway

Border Gateway (BG) is a router that provides a direct GPRS tunnel between the packet core networks of different operators via an inter-PLMN (Public Land Mobile Network) data network. When subscribers are visiting i.e. roaming in another PLMN, they can have connection to GGSN in their home PLMN via the BGs. The BG runs the standard BGP-4 (Border Gateway Protocol 4) routing protocol used to connect autonomous systems. The BG can also contain the firewall functionality. [4, p.6]

2.2.4 Network Management System

NetAct is Nokia Network Management System (NMS). In addition to the traditional NMS functions like alarm monitoring, configuration management and performance management, NetAct also supports the service configuration and service performance monitoring.

2.2.5 Lawful Interception Gateway

Lawful Interception Gateway (LIG) is essential network functionality within the GPRS and 3G infrastructures, providing legal authorities with the ability to intercept both GPRS and 3G data calls. Operators in most countries, among them all European Union member countries need to meet the local authority requirements before the commercial launch of GPRS and 3G network services. If the mobile subscriber is under interception, the 2G SGSN/3G SGSN and GGSN collects all related data call information and sends it to LIG. [5, p.11]

2.3 Nokia Intelligent Content Delivery system

Nokia Intelligent Content Delivery (ICD) is a service and content control system. It provides intelligent delivery of IP based content by enabling operators to analyse, charge and manage services. Examples of services that can be managed and charged with the ICD are browsing, streaming, downloading, Push to talk over Cellular (PoC), content-to-person multimedia messaging and person-to-person multimedia messaging.

For the subscribers ICD provides easy to use services. All services are accessible via a single Access Point (AP) with the same user equipment settings. Due to ICD online charging capabilities, all services can be provided to prepaid users without a fraud risk. For content and service providers ICD offers flexible revenue sharing models and secured delivery of premium content with confirmed content delivery. The network elements of the ICD system are shown in Figure 2.

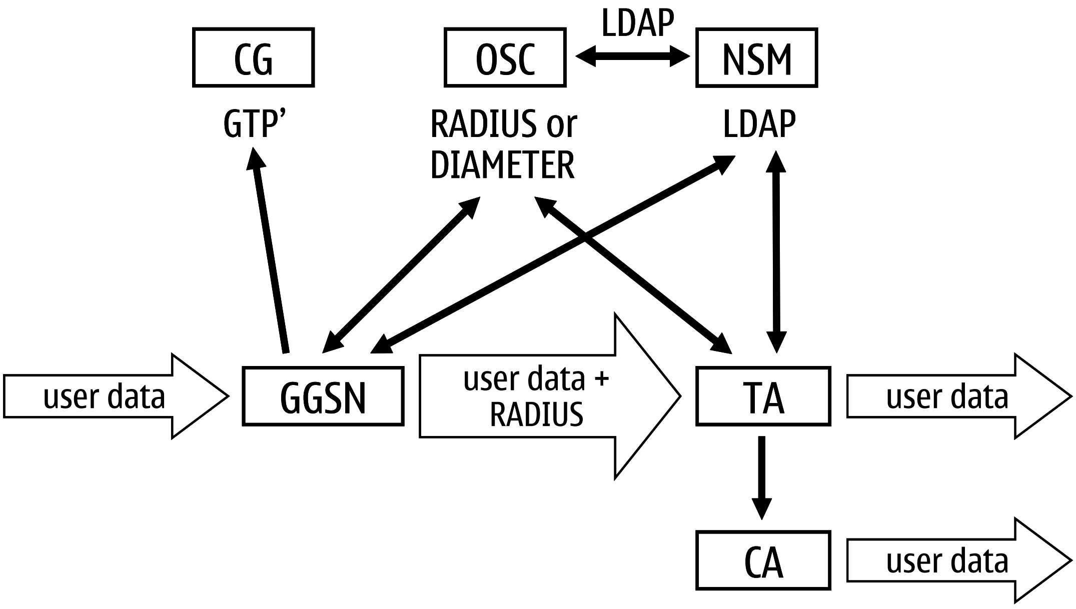

Figure 2 Nokia Intelligent Content Delivery system [6, p.15]

2.4 Network elements of ICD system

2.4.1 Nokia Subscription Manager

Nokia Subscription Manager (NSM) provides the service and subscriber information and works as a mediator towards the operator's Business Support System. The NSM is a management system to link mobile users, operator IP-based service offering, and the charging system. From the GGSN point of view, the main functions of NSM are: support for subscriber authentication, self-provisioning portal application, user and service profile database. The protocol used between GGSN and NSM is Lightweight Directory Access Protocol (LDAP).

2.4.2 Online Service Controller

Online Service Controller (OSC) enables online prepaid charging and service control for GPRS and 3G users. Main functionality covers charging event collection and logic, balance management and online service control. OSC collects charging events from the analysing elements (GGSN and TA), decides on the charging logic based on rules to be applied, synchronises and maintains the account balance in an external balance handling device and the services are barred or allowed as defined by rules and threshold based business logic.

GGSN and TA communicate with OSC to request quota for a set of services and to report usage of those services. The protocol used between GGSN and OSC is either RADIUS' (Remote Authentication Dial In User Service) or Diameter authentication, authorisation, accounting (AAA) protocol.

The OSC may receive rates for services from external rating systems. The OSC calculates the quota for GGSN based on the received rates and the user account balance. The OSC may also interface with subscription management systems for receiving Service Aware user data.

2.4.3 Charging Gateway

The 3rd Generation Partnership Project (3GPP) has specified Charging Gateway Function (CGF), which provides a mechanism for transferring charging information from the SGSNs and GGSNs to the network operator's post-processing system. In the Nokia implementation, CGF is implemented as a standalone element, Nokia Charging Gateway (CG). The CG is used in GPRS and 3G networks, as well as in Nokia ICD system. It consolidates raw event records into Charging Data Records (CDR) and forwards them in a suitable format to the post-processing system.

The main functions of CG are: collection of event records, intermediate event record storage buffering, processing of CDRs, and transfer of CDR data to the post-processing system. [7, p.41]

2.4.4 Gateway GPRS Support Node in ICD system

GGSN provides access to chargeable services and enforces online credit control. GGSN can interface with subscription management systems such as Nokia Subscription Manager (NSM) to retrieve user profiles containing the list of allowed services for each user. GGSN can also interface with service management systems such as ICD Service Directory (ISD) to retrieve service profiles containing service specifications. GGSN communicates with the Online Charging System to request quota for a set of services and to report usage of those services.

2.4.5 Traffic Analyser and Content Analyser

The Nokia Traffic Analyser (TA) and Content Analyser (CA) complement the GGSN by providing more in-depth charging capabilities and analysing capabilities. They also provide functionality for content handling and charging, as well as user interaction for advice-of-charge or notifications (for example, account exhaustion). The TA is used for traffic handling and charging features. The CA enables content charging and additional user control and interaction features.

2.5 Summary

In this chapter the position and the role of GGSN was explained. The basic role of GGSN in the packet core network is to operate as a gateway between the radio networks and the data networks. Through GGSN the end user can get access to the services provided by the mobile ISP, to Internet and to the corporate intranets.

In Nokia Intelligent Content Delivery system GGSN has central position. GGSN controls the end user access to the services in the data networks according the information it receives from the service and subscription management systems. GGSN also performs sophisticated traffic analysis and based on the analysis GGSN reports the usage of services to post-paid and prepaid charging systems.

GGSN has to communicate with many different types of neighbouring network elements. In the next chapter the functionality of GGSN is explained in more detailed and all used protocols are presented.

3. GATEWAY GPRS SUPPORT NODE

In this chapter the functionality of Gateway GPRS Support Node is explained in more details. The interfaces of the GGSN are presented and also information about all used protocols is provided.

3.1 Functionality of GGSN

The main purpose of the GGSN is to act as a gateway between the access networks and the public/private data networks (e.g. Internet and corporate intranets). The user-plane traffic consists normally of browsing e.g. HTTP (Hypertext Transfer Protocol) and WAP (Wireless Application Protocol), messaging e.g. e-mail and MMS (Multimedia Messaging Service) and streaming e.g. SIP (Session Initiation Protocol) and RTSP (Real-Time Streaming Protocol). One essential feature in the user-plane traffic processing is the traffic analysis. GGSN is able to classify traffic based on the Layer 4 (L4) and the Layer 7 (L7) attributes of the TCP/IP protocol stack. The L4 is the transport layer and the traffic analysing attributes are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) port numbers. The L7 is the application layer and the used attribute is URI (Uniform Resource Identifier). [8, p.17]

The other main functionality of GGSN is the signalling. It is divided into following dimensions:

- Signalling towards access networks. There is signalling, which is required for creating, modifying and deleting the PDP (Packet Data Protocol) contexts. The request for PDP context creation comes always from the external network element.
- Signalling towards data networks. Some of the signalling is required for configuring the PDP context. Most of this signalling happens when the PDP context is created. It is used for e.g. allocating the IP address for the User Equipment (UE).
- Charging. Mobile subscribers access services via GPRS and 3G networks, and they need to be charged. GGSN analyses the user-plane traffic and reports the measurement results to the charging system via

signalling interfaces. There are two types of charging interfaces: offline (post-paid) and online (prepaid) charging.

- Subscription management, authentication and session control. GGSN may need to authenticate the mobile subscribers before PDP context can be created. In addition, GGSN may need to know what services the mobile subscriber is allowed to use.
- Operation and maintenance. GGSN needs lots of configuration information. It may be configured locally, or there might also be a centralised network management system, which configures the GGSN via signalling interfaces. In addition, statistics are collected in GGSN and GGSN sends information about faults to external network management system.
- Lawful interception. GPRS network is public service, and in many countries the local authorities require possibility of monitoring the traffic for certain mobile subscribers. Lawful interception is controlled via signalling interfaces from external lawful interception system.

Following Figure 3 gives an overview of the logical architecture of the GGSN.

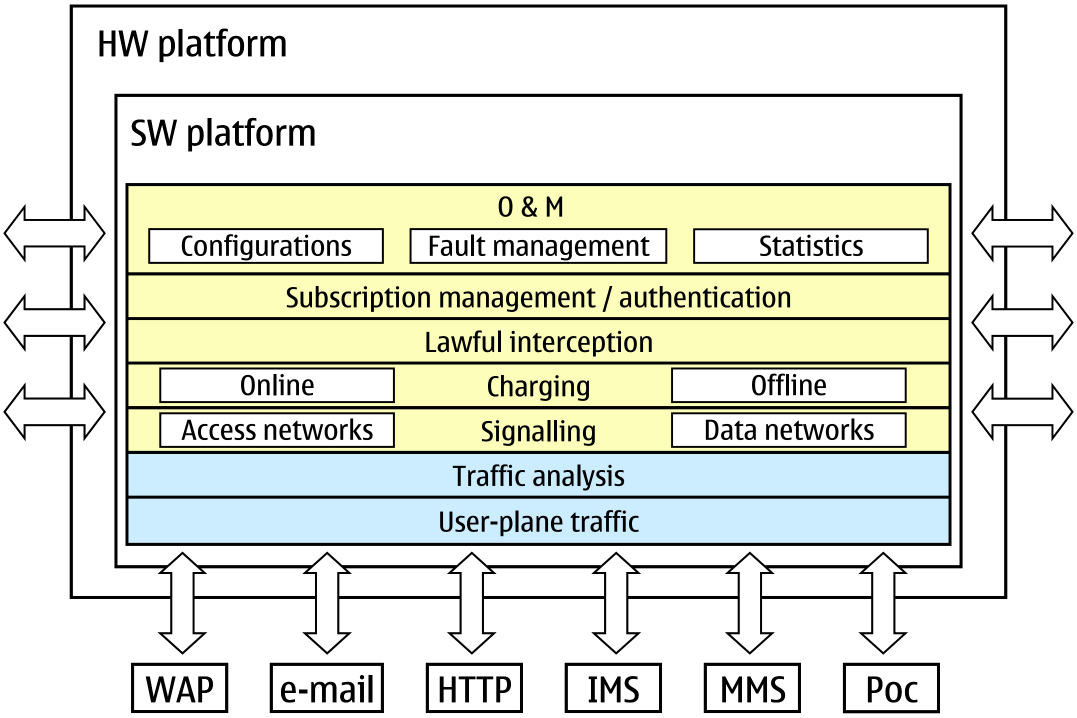

Figure 3 GGSN logical architecture [8, p.18]

3.2 GGSN interfaces, protocols and procedures

The interfaces of the GGSN are specified by 3GPP as depicted in Figure 4. Gn is the interface between GGSN and SGSN within the same PLMN. Gp is the interface between the GGSN of the Home PLMN (HPLMN) and SGSNs in Visited PLMNs (VPLMN). Gi is the interface between GGSN and external packet data networks. Ga is the interface between GGSN and Charging Gateway (CG).

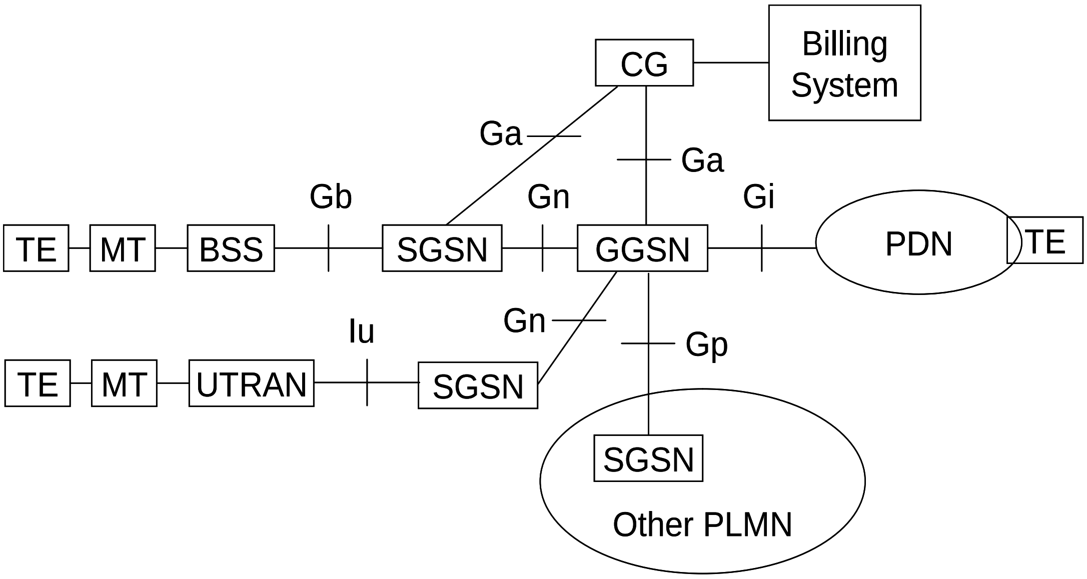

Figure 4 Interfaces of GGSN as specified by 3GPP [9, p.11]

3.2.1 Gn and Gp interface

The Gn and Gp interfaces are defined by 3GPP in the technical specification TS 29.060 [9]. The used protocol is GPRS Tunnelling Protocol (GTP). The GTP uses User Datagram Protocol (UDP) to carry user data over the Internet Protocol (IP). The GTP protocol stack is shown in Figure 5.

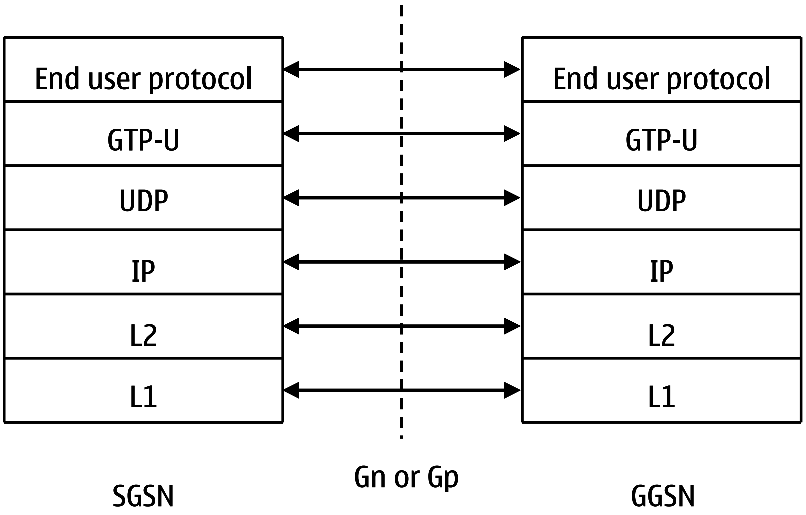

Figure 5 GTP protocol stack of Gn and Gp interfaces

There are two types of GTP namely GTP user plane (GTP-U) and GTP control plane (GTP-C). The GTP-U is used to carry PDP context data. The main function of GTP-C is creating, updating and deleting the PDP contexts.

3.2.2 Attach and detach procedure

Before a mobile station can use GPRS services, it must register with an SGSN. The network checks if the user is authorized, copies the user profile from the HLR (Home Location Register) to the SGSN, and assigns a Packet Temporary Mobile Subscriber Identity (P-TMSI) to the user. This procedure is called GPRS attach. After the GPRS attach the mobile station is logically connected to the SGSN. The disconnection from the GPRS network is called GPRS detach. It can be initiated by the mobile station or by the network (SGSN or HLR). The GGSN is not involved in attach and detach procedures. [10, p.97, p.108]

3.2.3 Session management, PDP context

To exchange data packets with external PDNs (Packet Data Network) after a successful GPRS attach, a mobile station must apply for the IP address used in the PDN. This address is called PDP address (Packet Data Protocol address). For each session, a so-called PDP context is created, which describes the characteristics of the session. It contains the PDP type (e.g. IPv4 or IPv6), the PDP address assigned to the mobile station, the requested QoS (Quality of Service), and the address of a GGSN that serves as the access point to the PDN. This context is stored in the MS, the SGSN and the GGSN. With an active PDP context, the mobile station is "visible" for the external PDN and is able to send and receive data packets. The mapping between the two addresses, PDP and IMSI (International Mobile Subscriber Identity), enables the GGSN to transfer data packets between PDN and MS. A user may have several simultaneous PDP contexts active at a given time. The allocation of the PDP address can be static or dynamic. In the first case, the network operator of the user's HPLMN permanently assigns a PDP address to the user. In the second case, a PDP address is assigned to the user upon activation of a PDP context. The PDP address can be assigned by the operator of the user's HPLMN or by the operator of the visited network. In case of dynamic PDP address assignment, the GGSN is responsible for the allocation and the activation/deactivation of the PDP addresses.

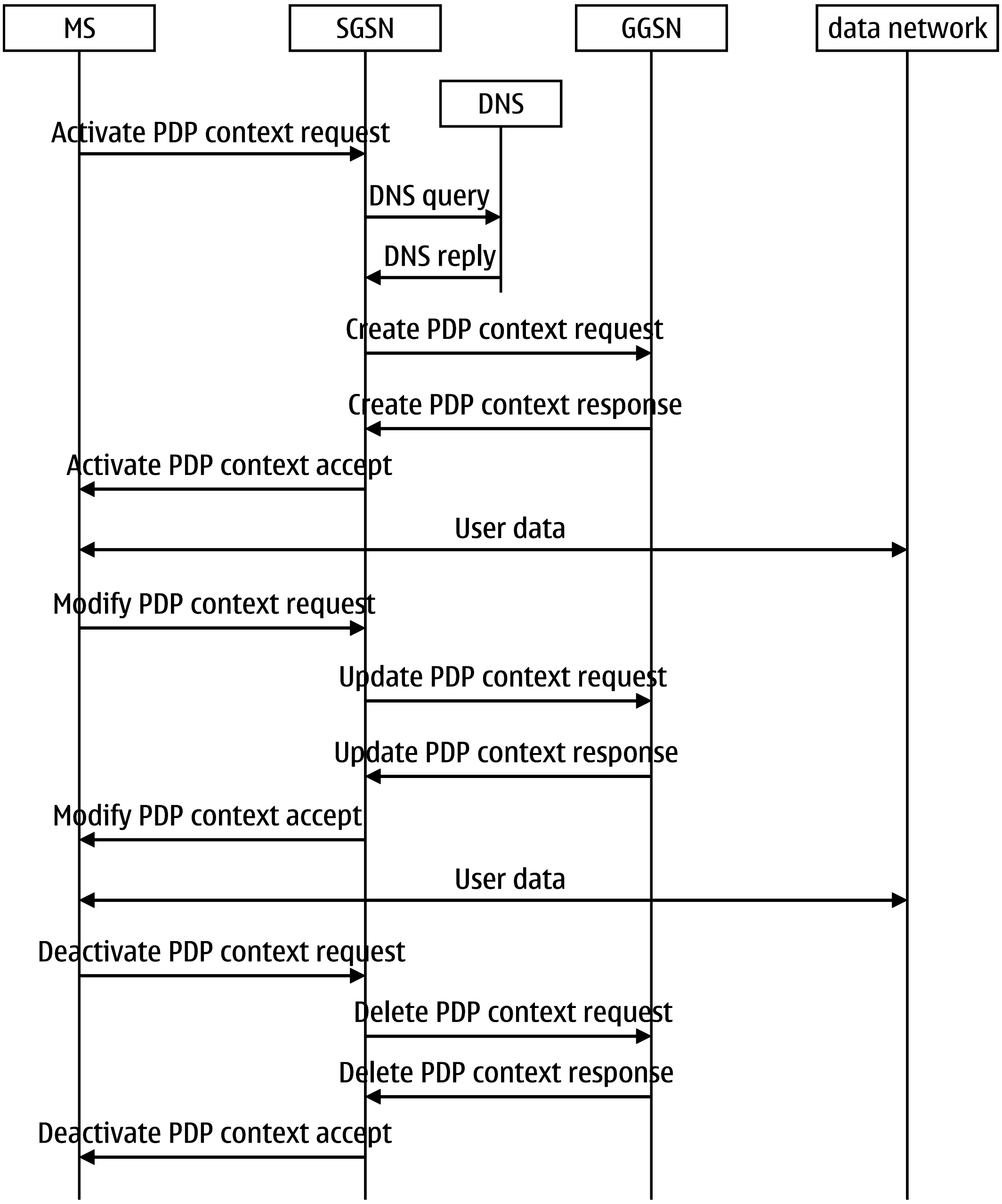
MS
SGSN
GGSN
data network
DNS
Activate PDP context request
DNS query
DNS reply
Create PDP context request
Create PDP context response
Activate PDP context accept
User data
Modify PDP context request
Update PDP context request
Update PDP context response
Modify PDP context accept
User data
Deactivate PDP context request
Delete PDP context request
Delete PDP context response
Deactivate PDP context accept

Figure 6 PDP context activation/deactivation procedure

Figure 6 shows the PDP context activation/deactivation procedure. Using the message "Activate PDP context request” the MS informs the SGSN that MS wants to connect to the definite PDN. In the before-mentioned request MS sends the Access Point Name (APN) to the SGSN. If dynamic PDP address assignment is requested, the parameter PDP address will be left empty. After this usual security functions (e.g. authentication of the user) are performed. If access is granted, the SGSN will solve the GGSN address by DNS (Domain Name Server) query and sends a "Create PDP context request" message to the GGSN where the requested Access Point (AP) exists. The GGSN creates a new entry in its PDP context table, which enables the GGSN to route data packets between the SGSN and the external PDN. Then the GGSN returns a confirmation message "Create PDP context response" to the SGSN, which contains the PDP address in case dynamic PDP address allocation was requested. The SGSN updates its PDP context tables and confirms the activation of the new PDP context to the MS ("Activate PDP context accept"). Now the PDP context is created and the user data can be transferred between MS and PDN.

If the MS wants to change the QoS parameters during an active PDP context, MS can send “Modify PDP context request” to SGSN. After this the SGSN sends “Update PDP context request” to GGSN. In addition to the PDP Context Modification procedure an “Update PDP Context Request” message shall be sent from a SGSN to a GGSN also as part of the GPRS Inter SGSN Routing Update procedure or to redistribute contexts due to load sharing.

The MS can delete the PDP context by sending “Deactivate PDP context request” message to SGSN. Then the SGSN send “Delete PDP context” message to GGSN. After this GGSN and SGSN confirms the PDP context deletion to MS. [9, pp.18-26]

3.2.4 Gi interface

The user plane traffic between GGSN and external packed data networks (e.g. Internet or intranets) can be plain IPv4 or plain IPv6 or the following encapsulation methods can be used: GRE tunnelling (Generic Routing Encapsulation), L2TP tunnelling (Layer Two Tunnelling Protocol) and IP-over-IP tunnelling.

3.2.4.1 GRE tunnelling

Generic Routing Encapsulation (GRE) is tunnelling protocol that is used to move logically a router interface to another place. GRE forms a tunnel from the real interface to a virtual one, operated by another router. GRE does not have security features therefore the data must be encrypted before it is encapsulated by the GRE. In GGSN, the IP packets are encapsulated as the payload of the GRE packet. The GRE packets are then delivered to the other end-point of the GRE tunnel over IP. The GRE tunnelling configuration is part of the Access Point configuration. The GRE tunnel is used to carry all the user-plane traffic. Optionally, the RADIUS and DHCP (Dynamic Host Configuration Protocol) signalling may also be carried over the GRE tunnel. GRE is an IETF (Internet Engineering Task Force) protocol defined by the RFC (Request For Comments) 1701 [11]. [8, p.56]

3.2.4.2 IP-over-IP tunnelling

In IP tunnelling, the actual IP packets are encapsulated inside another IP packet as payload. The new IP packet, which now contains the original IP packet, is sent to the other end-point of the IP tunnel. There the IP packet is decapsulated, yielding the original IP packet, which is then delivered to the destination indicated by the destination address field of the original IP packet. Besides the basic encapsulation where another IP header is added to the original IP packet, the IP tunnels supports also IP authentication header. The IP tunnels are used also for cases, where the IPv6 packets need to be delivered over IPv4 networks. IP-over-IP is an IETF protocol defined by the RFC 2003 [12]. [8, p.59]

3.2.4.3 L2TP tunnelling

Point-to-Point Protocol (PPP) frames can be encapsulated and carried from the GPRS/3G network to a corporate network by using the Layer Two Tunnelling Protocol (L2TP). In the L2TP tunnel, an L2TP Access Concentrator (LAC) and an L2TP Network Server (LNS) act as endpoints of the tunnel. The GGSN implements the LAC side of the tunnel. The LNS is required as an endpoint in the corporate network. For every PDP context with its own IP address, a separate L2TP session is established within the tunnel. The L2TP uses two types of messages: control messages and data messages. The control messages are used in the establishment, maintenance and clearing of tunnels and calls. The data messages are used to encapsulate the PPP frames being carried over the L2TP tunnel. The L2TP does not provide its own security, but it can make use of IPsec (Internet Protocol security). The GGSN supports the following PPP protocols in a virtual PPP session: Link Control Protocol (LCP), IP Control Protocol (IPCP), Password Authentication Protocol (PAP) and Challenge

Handshake Authentication Protocol (CHAP). The LCP is the basic negotiation protocol in PPP. The IPCP is used to configure the IP parameters of the PPP session. It can be used to allocate IP address and DNS servers to the mobile terminal. The PAP and the CHAP are used for authentication. Authentication is based on PAP authentication parameters received from the mobile terminal, or alternatively, the authentication can use the IMSI, MSISDN (Mobile Subscriber International ISDN Number) or access point name. The DHCP and RADIUS signalling is not required when L2TP tunnelling is used, because the IPCP and PAP/CHAP are used. L2TP is an IETF protocol defined by the RFC 2661 [13]. [14, pp.22-25]

3.2.4.4 User Authentication with RADIUS

Nokia GGSN supports user authentication using RADIUS. As the GGSN cannot make queries to the MS, all required information must be relayed through the SGSN in the initial PDP context activation request.

The MS can request your laptop for a user name and a password at the PPP (Point-to-Point Protocol) connection set-up. This information is sent from the SGSN to the GGSN inside the PDP configuration options of the activation request message. The SGSN does not touch these information elements – it is just relaying the data back and forth.

A RADIUS server is normally used with dial-up connections. A GGSN is not able to make use of all features of a RADIUS server because the protocol between the GSNs (GPRS Support Nodes) does not support them. Most notably, the GGSN does not deliver any RADIUS attributes to an MS apart from an IP address. A RADIUS server is typically located in an external network, although technically it could reside in the internal GPRS backbone too.

If the selected Access Point includes user authentication and/or IP address configuration via RADIUS, the GGSN sends an Access-Request message to a pre-configured RADIUS server IP address with the user name and password combination. The RADIUS server checks the information against the user database and either accepts or rejects the request. A positive response may include an IP address. RADIUS is an IETF protocol defined by the RFC 2865 [15]. [16, p.41], [17, module 4 p.5]

3.2.4.5 Dynamic IP Address Allocation

In addition to RADIUS, the Nokia GGSN software can allocate dynamic IP addresses with two other methods: using an Access Point specific address pool or from a DHCP server.

The GGSN can dynamically reserve addresses from a local address pool. The addresses in use will be released back into the pool at PDP context deactivation. This method is fast and simple and technically the soundest.

The alternative to the GGSN local address pool is using DHCP. It is best suited for corporate intranets already using DHCP for distributing TCP/IP parameters to personal computers.

When the AP connects to such a network, the GGSN sends a DHCP Discover message to the IP address of a pre-configured DHCP server. The DHCP server makes a reservation for an IP address and returns it, along with other TCP/IP parameters, in a DHCP Offer message. The GGSN accepts it with a DHCP Request and receives a DHCP Ack as an acknowledgement.

Finally, if all checks pass and GGSN obtains a dynamic address successfully, it acknowledges the PDP context activation request to the SGSN, whereupon the SGSN does the same towards the MS. DHCP is an IETF protocol defined by the RFC 2131 [18]. [16, p.34], [17, module 4 p.6]

3.2.5 Ga interface

Charging Data Records (CDRs) are sent from the GGSN to the Charging Gateway (CG) using Enhanced GPRS Tunnelling Protocol (GTP’). The CDR includes fields identifying the user, the session and the network elements as well as information on the network resources and services used to support a subscriber session. The messages used by the GTP’ protocol are shown in Table 1.

The GGSN supports two types of CDRs. The normal charging is based on G-CDR (GGSN Charging Data Record). The G-CDR contains the measured values of amount of transferred uplink and downlink user traffic and duration of the PDP context i.e. G-CDR is the CDR, which contains data related to the whole PDP context.

Another CDR type is SA-CDR (Service Aware Charging Data Record). By using the SA-CDR the user can be charged in more detail according the usage of the services. Each service can be charged separately. The usage of the services can be charged based on the uplink or downlink volume of the user traffic, or based on the time the service is used, or based on the hit counts to the specified URIs (Uniform Resource Identifier).

Table 1 GTP’ messages [19, p.44]

Message	**Purpose of use**
Node Alive Request	Message is used to inform that the GGSN or the CG is available for service.
Node Alive Response	Message is sent as a response to a received Node Alive Request.
Redirection Request	The CG sends this message to advise that received CDR traffic is to be redirected to another CG due to that CG node is about to stop service (due to an outage for maintenance or an error condition).
Redirection Response	With this message the GGSN responses to a received Redirection Request.
Version not Supported	The CG answers with this message if it receives a CDR which version the CG does not support. This message is also sent from the GGSN to the CG as a response to a Node Alive Request or a Redirection Request when GGSN doesn't support the GTP' version of the packet.
Data Record Transfer Request	Message is used to transmit the CDR information, which is placed in the Data Record information element.
Data Record Transfer Response	Message is sent as a response to received data Record Transfer Request.

3.2.6 NSM interface

When the user equipment activates a PDP context to access a certain service, the GGSN retrieves the user profile from the NSM using Lightweight Directory Access Protocol (LDAP) as shown Figure 7. LDAP is an IETF protocol defined by nine different RFCs. These RFCs are listed in the RFC 3377 [20, p.1].

The user profile includes the following information: *User identification* (IMSI or MSISDN (Mobile Subscriber International ISDN Number)), *Charging profile* of the user (post-paid user or prepaid user) and *Allowed services* for the user.

Each allowed service has the following parameters:

- *Service name*

- *Authentication* (username and password). This parameter can be used for service authentication. The operator can store the username/password combination in the user profile. This pair is then used for service-originated authentication. In that case, the mobile user does not need to know the actual authentication information.

- *Activation Status:* active/inactive/default. By this activation status the operator can allow or deny the usage of certain services. This functionality can be combined with the online prepaid functionality in order to block certain services when the user's prepaid balance is zero. It is also possible to update the service information during an active PDP context.

3.2.7 OSC interface

The OSC interface is used to enable the online prepaid charging for GPRS/3G subscribers. The messages between GGSN and OSC are depicted in Figure 7.

After the GGSN has received the "Create PDP context request" from the SGSN, the GGSN fetches the User profile from NSM. In the User profile the GGSN gets information of the allowed services for the subscriber. In GGSN there are configured Charging classes for each service. The GGSN sends an "Accounting request (start)" message towards the OSC, with the

Charging class list. The OSC receives the message and request rating information for the Charging classes from internal or external Rating engine. With the rating information OSC will calculate the thresholds/tokens and send them back to the GGSN. After this the GGSN starts to count user data packets.

When the available thresholds/tokens are consumed or configured maximum time limit is reached, the GGSN will report the usage of services to the OSC by sending the “Accounting request (interim update)” message to the OSC. Then the OSC will answer by sending again the thresholds/tokens in the “Accounting response (interim update)” message to the GGSN if there is still money left in the subscriber’s prepaid user account. The GGSN will also send the “Accounting request (interim update)” message to the OSC if the PDP context is updated (because of QoS change or SGSN change) or the GGSN receives an “Update user profile” message from the NSM (i.e. the allowed services for the subscriber are added or deleted).

When the subscriber deactivates the PDP context, the SGSN sends the “Delete PDP context request” message to the GGSN and the GGSN reports the used services to the OSC by the “Accounting request (stop)” message. The OSC replies with the “Accounting response (stop)” message and the PDP context will be deleted. [21, pp.16-19]

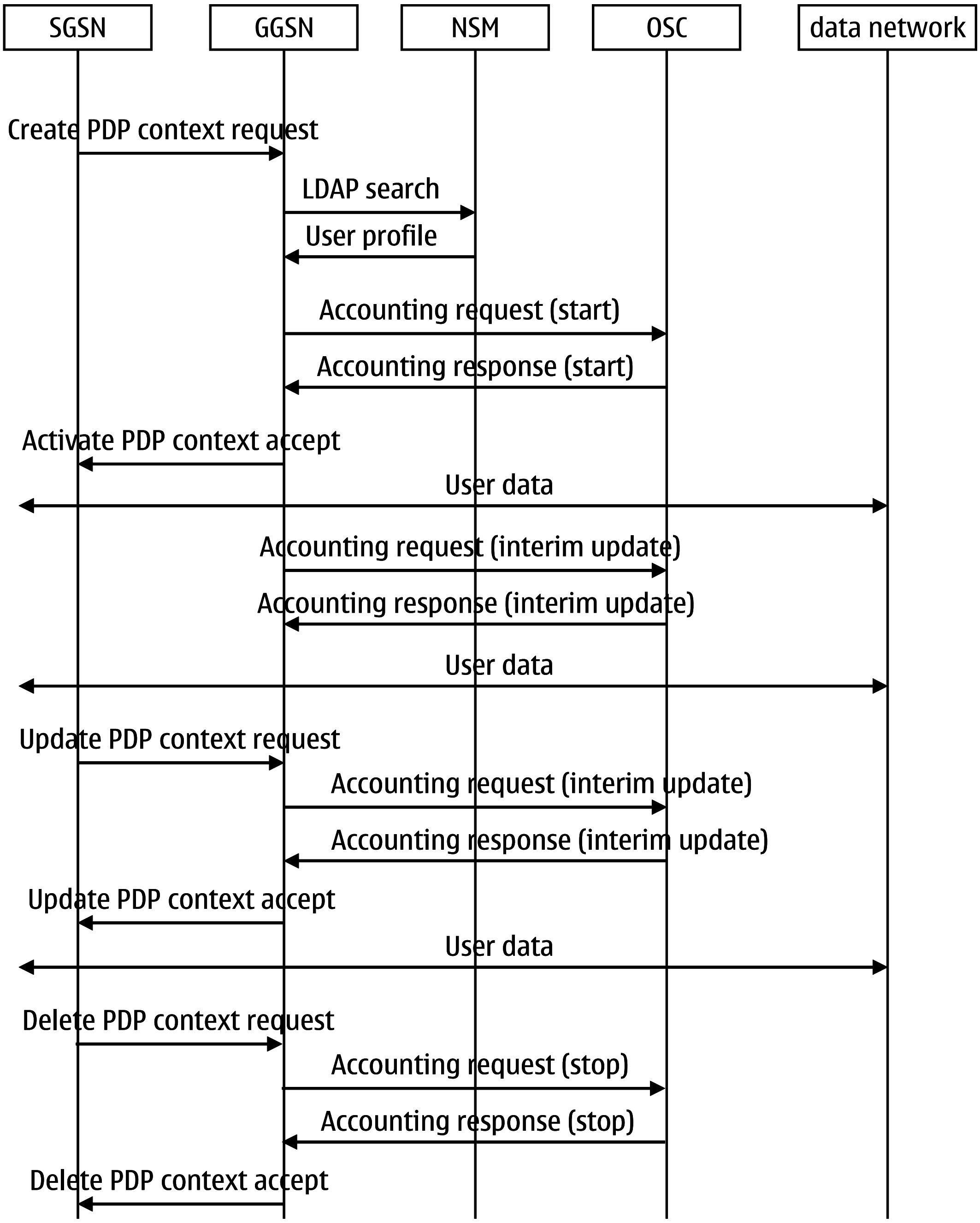
SGSN
GGSN
NSM
OSC
data network
Create PDP context request
LDAP search
User profile
Accounting request (start)
Accounting response (start)
Activate PDP context accept
User data
Accounting request (interim update)
Accounting response (interim update)
User data
Update PDP context request
Accounting request (interim update)
Accounting response (interim update)
Update PDP context accept
User data
Delete PDP context request
Accounting request (stop)
Accounting response (stop)
Delete PDP context accept

Figure 7 User profile fetching and online prepaid charging

3.2.8 Network management interface

The functions of the network management interface are fault management, performance management and configuration management of the network element. The protocols used in the GGSN network management interface are: HTTP (Hypertext Transfer Protocol), Telnet (Terminal emulation protocol), SNMP (Simple Network Management Protocol), NE3S (Nokia Enhanced SNMP Solution Suite), FTP (File Transfer Protocol), NTP

(Network Time Protocol) and LDAP (Lightweight Directory Access Protocol).

The main user interface in GGSN is based on the client-server model and it uses HTTP protocol. Platform of GGSN contains a web server named Voyager and any web browser acts as a remote client application to configure and monitor the network element. The GGSN can be configured also by using a Command Line Interface (CLI). The CLI uses the Telnet protocol. Alternatively, the configuring and monitoring of the GGSN can be done through the centralised NetAct management system. The NetAct uses SNMP protocol for reading and writing parameters controlling the operation of GGSN by issuing SNMP GetRequest, GetNextRequest and SetRequest messages. The Get messages will be replied with GetResponse message from the GGSN.

The GGSN can set and cancel alarms for special problem situations. The alarms can be browsed and cancelled on the GGSN Voyager management web pages by using a web browser. Alternatively, this can be done through the NetAct management system. The fault management towards the NetAct is based on SNMP traps the GGSN is able to send when an error occurs or it is cleared. There is one trap to set an alarm and another one to cancel it.

A standard FTP interface is provided to allow file transfer to/from the GGSN. The FTP interface can be used for the GGSN SW package downloading or for transferring the GGSN backup and log files.

The GGSN provides statistical data for performance management purposes. The GGSN MIB (Management Information Base) defines what statistics are supported. The measurement jobs of performance indicators are defined in measurement schedule file. NetAct creates this file and transfers it to the GGSN by using FTP protocol. The NetAct activates a measurement job by setting a corresponding SNMP object to active state in the GGSN by using SNMP SetRequest message. The GGSN executes the measurement job defined in the schedule file and puts the measurement results into a result file and informs the NetAct that the measurement is ready by sending the notification to the NetAct. The NetAct retrieves the result file from the GGSN by using FTP protocol. The NetAct can also directly read the performance indicators by using SNMP protocol.

The Network Time Protocol (NTP) is used for synchronising the GGSN internal clock. [22, pp.6-25]

In the ICD system the service profiles can be stored in the central repository called ISD (ICD Service Directory). If the interface towards the ISD is enabled in the GGSN, the GGSN regularly polls the ISD for service changes and applies the changes locally. The service profile received from the ISD will always override the local configuration in the GGSN. The service profiles are fetched using the LDAP protocol. At regular intervals the GGSN will also provision some data to the ISD. The GGSN updates part of the access point configuration to the ISD. [23, pp.4-8]

3.3 Summary

This chapter presented the architecture of GGSN. All interfaces of GGSN and used protocols are explained. The traffic from the mobile user comes to GGSN via SGSN by using GPRS Tunnelling Protocol. In GGSN this GPRS Tunnelling Protocol is decapsulated and the traffic is forwarded towards data networks by using either plain IPv4 or plain IPv6 protocol, or by one of the following tunnelling methods: GRE tunnelling, L2TP tunnelling or IP over IP tunnelling. The user and service profiles are fetched from NSM and ISD by using LDAP protocol. The offline (post-paid) and online (prepaid) charging information is reported to charging systems by using GTP’ or Radius’/Diameter protocols.

4. SOFTWARE TESTING

This chapter gives general information about the software testing and software development process. A risk based testing method is described and different testing phases are introduced. Also the importance and the benefits of early involvement of testing are explained.

4.1 Definition of software testing

The definition of software testing has changed over the years. In 1979, Glenford Myers explained in his book, The Art of Software Testing [24, p.5]: “Testing is the process of executing a program with the intent of finding errors.”

In 1983, Bill Hetzel stated in his book, Complete Guide to Software Testing [25, p.6]: “Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results.”

In 2002, Rick Graig and Stefan Jaskiel defined the software testing in their book, Systematic Software Testing [26, p.4]: “Testing is a concurrent lifecycle process of engineering, using, and maintaining testware in order to measure and improve the quality of the software being tested.”

So, the scope of software testing has been broadening over the years. In the 1970s the software testing occurred at the end of the software development cycle and the main purpose was to find defects. In the 1980s the definition of testing had changed to include an assessment of the quality of the software, rather than merely a process to find defects. Nowadays the aim is that testing is involved already in the beginning of the software lifecycle by participating into inspections and reviews and this way testing can also prevent defects and improve quality i.e. the emphasis should be on defect prevention rather than defect removal.

When testing a program, the aim is to prove that the program doesn’t work. The search is not for tests that the program can pass. The search is for tests that the program will fail.

E.W.Dijkstra [27, p.85] argued: “Program testing can be used to show the presence of bugs, but never to show their absence!”

Quite often erroneously the approach for testing is to show that a system does what it should and doesn’t do what it shouldn’t, i.e. the goal of testing is to show that the system works. This is an impossible task when testing any non-trivial system. Therefore this kind of goal does not motivate testing personnel and the likelihood of finding errors will be decreased. A better testing approach is to show that the system does what it shouldn’t and doesn’t do what it should, i.e. the goal of testing is to find errors and the testing is successful if the system fails.

4.2 Risk based testing

The purpose of risk based testing is to concentrate to the features that are most important to the customers and to the features that most probably have defects. The principle of the risk based testing is depicted in Figure 8.

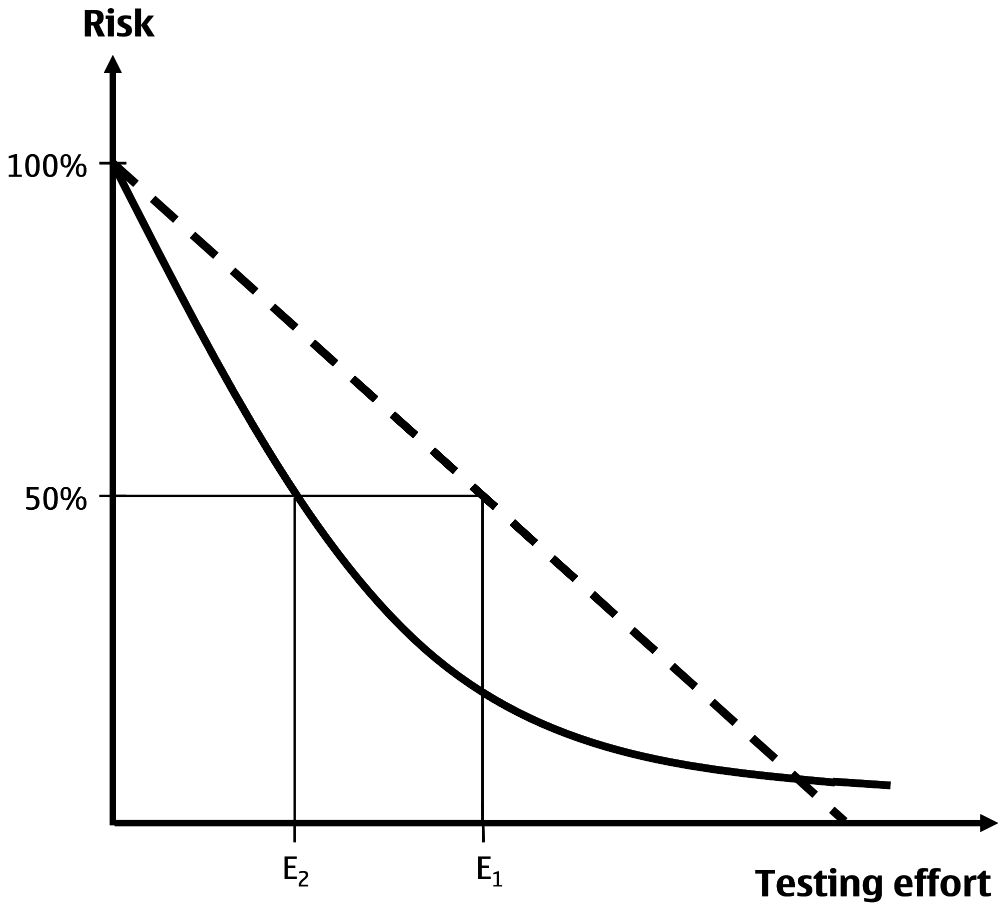

Figure 8 Traditional testing vs. risk based testing

In Figure 8 the dashed line shows the situation where each feature is equally important to test. Testing effort needed to mitigate the risk to 50% is E_1. In a real-life situation some features are more important to customers and for some features the probability of defects is higher than for other features. Therefore by concentrating into the correct features testing effort needed to mitigate the risk to 50% is E_2 that is substantially less than E_1.

In the risk based testing all the features of the software must be analysed. The purpose of a software risk analysis is to determine what to test, the testing priority and the depth of testing.

The software risk analysis can be done as shown in Table 2.

Table 2 An example of software risk analysis

Step 1: List all features of the product in the table and estimate the likelihood and impact of failure			
Feature	Likelihood of failure 1 = very high 2 = high 3 = medium 4 = low 5 = very low	Impact on the user 1 = severe 2 = major 3 = moderate 4 = minor 5 = negligible	Risk priority = Likelihood x Impact
u	1	1	1
v	5	2	10
w	2	4	8
x	1	5	5
y	3	4	12
z	2	3	6

Step 2: Sort the table according to the risk priority			
Feature	Likelihood of failure	Impact on the user	Risk priority = Likelihood x Impact
u	1	1	1
x	1	5	5
z	2	3	6
w	2	4	8
v	5	2	10
y	3	4	12

Step 3: Decide what has to be tested, what will be tested if there is time and what will not be tested			
Feature	Likelihood of failure	Impact on the user	Risk priority = Likelihood x Impact
u	1	1	1
x	1	5	5
z	2	3	6
w	2	4	8
v	5	2	10
y	3	4	12

In the first step all features of the software are listed, then for each feature the likelihood that the feature will fail to operate correctly is estimated. When the likelihood is estimated, the following issues should be taken into account: What features have had defects in earlier software releases? From what features the earlier test phases have found defects and from what features the customers have found defects? New or modified features will have more defects. Complex features will have more defects.

After the likelihood estimation the impact on the user if the feature will fail is also estimated. Then the risk priority for each feature is calculated by multiplying likelihood by impact.

In Step 2 of Table 2 the features are sorted according to the risk priority.

In Step 3 it must be decided where to draw the line i.e. what will be tested thoroughly and what will be tested partially and what will not be tested at all.

It is also important that the results of the risk analysis may have to be reviewed occasionally during the test planning and test execution phase since the requirements, resources, and other factors may change. [26, pp.28-47]

4.3 Software lifecycle model

For the GGSN SW projects the V lifecycle model is used as the SW development process. The V-model is depicted in Figure 9. The GGSN is developed as a product program. The phases of the product program are illustrated in the lower part of Figure 9. The V-model proceeds from left to right, depicting the basic sequence of development and testing activities. The left-hand side of the “V” detailing requirements, design, specification and coding phases, and the right-hand side detailing the integration and verification phases. Each phase has entry and exit criteria, which must be satisfied before the next phase commences. The GGSN is part of the larger system e.g. mobile packet core or ICD system. These larger systems are developed as system programs. The phases of the system program are shown in the upper part of Figure 9.

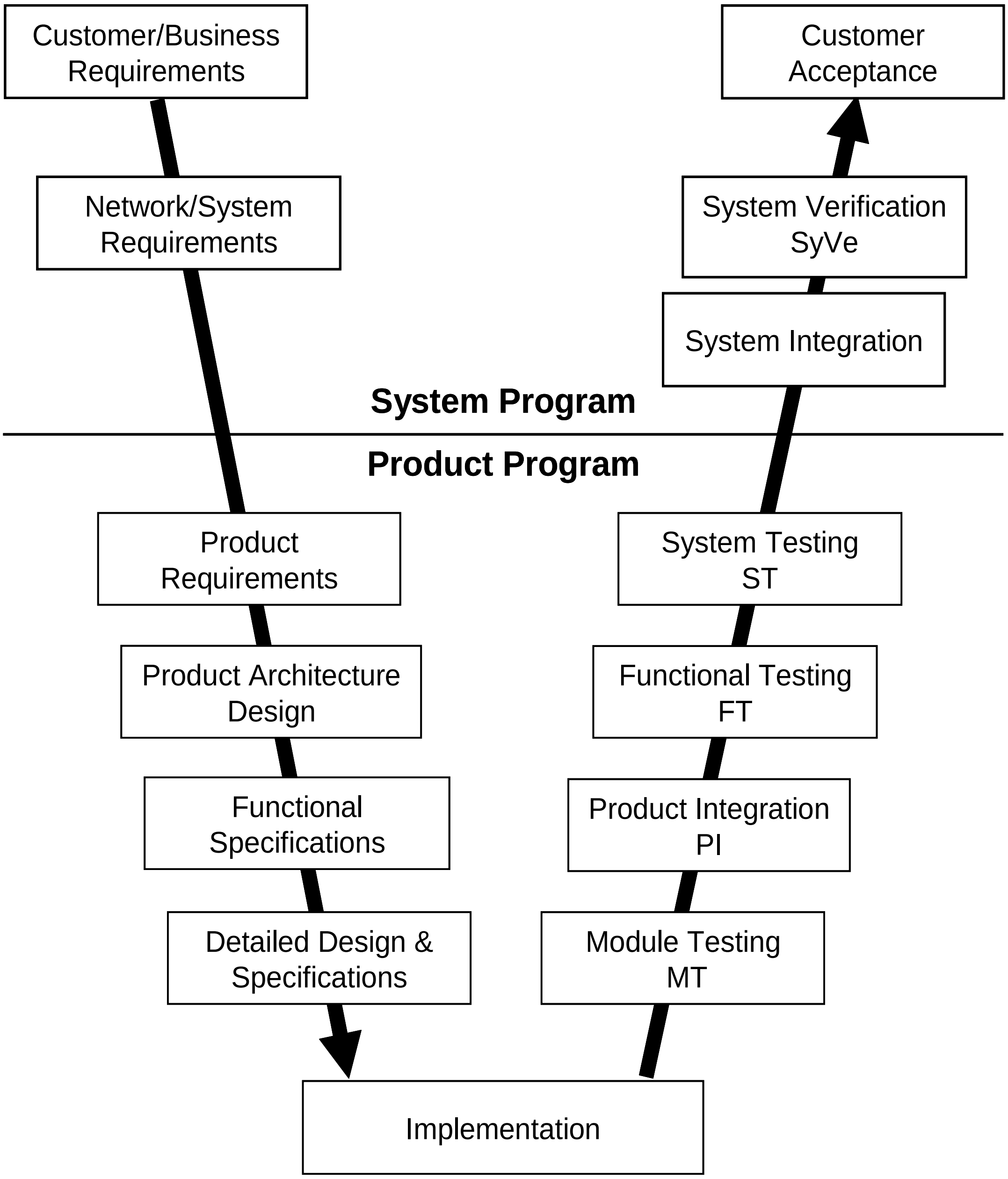

Figure 9 V lifecycle model for SW development

4.4 Test phases

The software testing can be divided in the following test phases: Module testing, Product integration, Functional testing, System testing, System verification and Acceptance testing. Each test phase can be divided into the following steps: test project planning, test planning, test design, test execution and reporting.

4.4.1 Module Testing

In Module Testing (MT) each SW module is tested and debugged separately. The design team performs Module Testing. The approach of the Module Testing is white box testing i.e. the designer knows the software structure in detail. The aim is to find module level bugs.

4.4.2 Product Integration

The purpose of Product Integration (PI) is to assemble the product from the product components, and to ensure that the complete product, as integrated, functions properly. Product Integration is usually done in incremental stages. Product Integration activities include preparation, assembly and integration test. The design team perform the Product Integration in co-operation with the testing team.

4.4.3 Functional Testing

The purpose of Functional Testing (FT) is to find functional level bugs and to ensure that the implementation of the features conforms to the implementation and requirement specification of the feature, and to ensure that features work together in the way they are designed.

In FT the product is seen as a grey box. This means that the knowledge about product internal structures and which blocks are required to perform a feature and how blocks interact with each other is available. An independent testing team performs the Functional Testing.

4.4.4 System Testing

System Testing (ST) is the last test level of the product program before the SW and HW are delivered to a customer, which can be internal (e.g. System Verification in System Programs) or external. The purpose of ST is to ensure that the whole product (system) works as it is meant to work in the circumstances and procedures that correspond to the intended use of the product. ST gains confidence in the product that it is ready for delivery to the customer. In ST the product is seen as a black box without assuming any deeper or detailed knowledge about the implementation.

In System Testing the tests are based on requirement specifications, operator use cases and standards whereas in Functional Testing the tests are based on product architecture design, functional specifications and individual feature specifications.

System Testing can be divided into the following subareas: Release Upgrade Testing (RUGT), Performance Testing (PET), Compatibility Testing (CT) and Release Testing (RT).

4.4.4.1 Release Upgrade Testing

The target is to validate that the product in an older version can be upgraded as expected to the version to be released. For new products that do not have older versions, installation testing replaces upgrade testing.

4.4.4.2 Performance Testing

Performance Testing can be divided into several subareas. The most important subareas are as follows: Comparison testing compares the performance of new and previous SW releases. Stability testing verifies that the network element functions correctly with at least such traffic intensity that it is normally designed to handle and that the network element is stable when long time loaded. Overload testing verifies that the network element stays stable, maintains its ability to handle the nominal traffic when overloaded. Provocative testing verifies that the network element can handle normal maintenance and recovery procedures, can handle abnormal situations and redundancy methods works as specified. Capacity measurement finds the processes and events that require most processing time for dimensioning purposes and optimisation.

4.4.4.3 Compatibility Testing

The purpose is to validate the compatibility of the product in itself and towards other products. Compatibility of the product in itself covers two areas. Firstly, all parts in all allowed combinations of a product release fit together. Secondly, this product release is compatible with former product releases.

4.4.4.4 Release Testing

The purpose is to validate that new release is ready for customer delivery. Sometimes this may require customer specific settings and configuration in test environment. The customer or a representative in Customer Services organisation typically selects test cases and those cases may be also used in Customer acceptance testing.

4.4.5 System Verification

The target of System Verification (SyVe) is to verify end-to-end functionality in real environment and with real applications. The real environment means that there are all network elements and real mobiles/terminals.

4.4.6 Acceptance Testing

Acceptance Testing is usually performed under controlled conditions at the customer site, and the testing therefore matches the real-life behaviour of the system. The customer is in a position to see the product in operation and to verify that its behaviour and performance are as agreed and meet the customer requirements.

4.5 E-milestones of software process

As depicted in Figure 9 the software development can be divided into several design and testing phases. Before the project can proceed from one phase to another the deliverables of the project must fulfil set targets. In Figure 10 is presented the project milestones B2…E10. These are checkpoints for the system maturity.

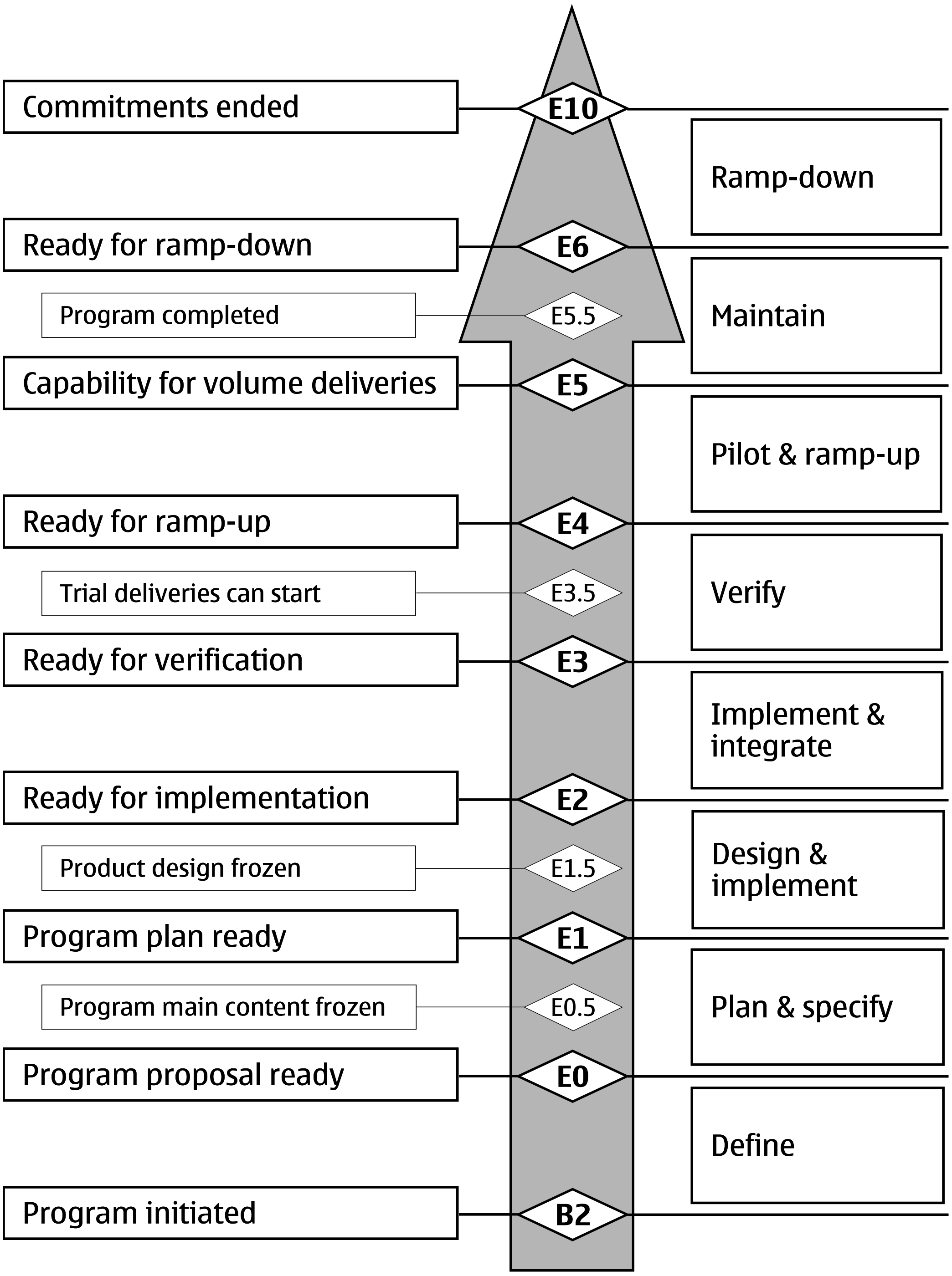

Figure 10 E- milestones of software process

The decision, whether the project meets the milestone criterias, is done in a project milestone review. It is a formal meeting at which a document or other product is presented to interested parties for approval.

Functional testing starts at E3 and ends at E3.5, System testing starts at E3.5 and ends at E4.

4.6 Error, fault and failure

The terms error, fault and failure can be defined as follows. A person makes an error (mistake), which creates a defect (fault, bug) in the software, which can cause a failure in operation. It is important to realise that some of the defects in the software do not ever cause a failure in the operation.

4.7 Cost of a defect and early involvement

The earlier in the software development process the defect is discovered, the cheaper it will be to correct it. Table 3 and Table 4 outlines the relative cost to correct a defect depending on the software lifecycle phase in which the defect is discovered. In both of those two estimates it can be seen that the correction cost increases very rapidly over the software lifecycle phases. So, the most cost efficient testing method is error prevention. It avoids the propagation of defects to later development phases. Therefore testers and other stakeholders should be involved from the beginning of the software lifecycle. E.g. the testers should participate into the requirement specification reviews. This way the testers can help to recognise omissions, discrepancies and other problems in the product requirements. Also the low level tests should be emphasised, because it is more cost efficient if the developer detects a defect than an independent testing team. The developer does not have to create a defect report and it is not necessary to communicate to anyone about the detected error. It is just enough if the developer corrects the defect before the software is delivered to the later testing phases.

Table 3 Relative cost to correct a defect, estimate 1 [28, p.4]

Phase where defect is discovered	**Relative cost to correct a defect**
Definition	1
High-level design	2
Low-level design	5
Code	10
Unit test	15
Integration test	22
System test	50
Post delivery	100+

Table 4 Relative cost to correct a defect, estimate 2 [29, p.27]

Phase where defect is discovered	**Relative cost to correct a defect**
Requirements	1
Design	3-6
Coding	10
Development testing	15-40
Acceptance testing	30-70
Production	40-1000

Other important aspects of early involvement of the testers are: “Testers need a solid understanding of the product so they can devise better and more complete test plans, designs, procedures, and cases. Early test-team involvement can eliminate confusion about functional behaviour later in the project lifecycle. In addition, early involvement allows the test team to learn over time which aspects of the application are the most critical to the end user and which are the highest-risk elements. This knowledge enables testers to focus on the most important parts of the application first, avoiding over-testing rarely used areas and under-testing the more important ones.” [28, p.3]

4.8 Independent testing

The purpose of independent testing is to provide a different perspective and, therefore different tests; furthermore, to conduct those tests in a richer (and, therefore more complex and expensive) environment than is possible for the developer. The purpose of the tests performed by the developer is to eliminate those defects that can be found at lower cost in the simpler, deterministic, environment of the module testing. [30, p.13]

4.9 Completion Criteria

In theory the decision-making when to stop the testing is a trivial task:

Risk of stopping testing = Risk of continuing testing

But estimating the risk is the difficult task. In practise there are a number of different ways to determine the test phase of the software lifecycle is complete. Some common examples are: White-box test coverage targets are met. Requirement test coverage targets are met. All test cases are executed. Rate of fault discovery goes below a target value. Measured reliability of the system achieves the target value (mean time to failure). Test phase time or resources are exhausted.

4.10 Summary

This chapter presented some thoughts about the software testing in general. Conventionally the main purpose of testing is to find the bugs and verify that the bugs are corrected, but much more cost efficient ambition for testing is error prevention. Some of the errors can be prevented by inspecting and reviewing the requirement and design specifications. Also the testing department should participate in the inspections and reviews. The error prevention can save significant amount of time and effort in the development and in the testing.

The basics of the risk based testing are also introduced in this chapter. The risk based testing concentrates on the most critical and the most important functionality areas of the product. This way the efficiency of the software testing can be increased and the quality of the product will be higher.

The software development process used in GGSN projects is the V-model. This software lifecycle model highlights the existence of different levels of testing and illustrates the way each relates to a different development phase. The milestones are as control gates which the project deliverables must pass before proceeding to the next lifecycle phase.

Even though the error prevention is so efficient method, this thesis concentrates mainly on the traditional testing method i.e. the error detection, because the error detection has been the main testing method in GGSN projects. The purpose of this study is to analyse GGSN testing and in the prevention side there are not much data to analyse yet. Following chapters present GGSN testing process analysis. The first part of the analysis is GGSN testing metrics which are illustrated in the following chapter.

5. TESTING METRICS

This chapter explains the software testing metrics. Some of the GGSN testing metrics which are collected during the testing process analysis are presented. Examples of the following testing metrics groups are provided: testing effort metrics, defect metrics, test case metrics, testing effectiveness metrics.

5.1 Testing metrics overview

Testing metrics are very valuable in test planning. Using history testing metrics data improves the test planning accuracy. It is much easier to estimate the needed time and resources for each test phase when metrics of earlier projects are available. The metrics provide a sound basis for future estimates.

During the testing metrics can be used to decide the optimal time of entry and exit for each test phase. Finally the product maturity can be estimated and release decision can be made by using the testing metrics.

Also the effectiveness and efficiency of testing can be evaluated. Testing efficiency is a productivity metric and testing effectiveness is a quality metric. Hence the testing efficiency metric can be e.g. "Number of test cases executed per hour”, and the test effectiveness metric can be e.g. “Number of defects discovered in the definite test phase per total number of defects discovered in all test phases”.

The testing metrics are also specifically important for improving the test process to assess the consequences for certain improvement measures, by comparing data before and after the implementation of the measure.

A risk with any metrics activity is dysfunctional measurement, in which participants alter their behaviour to optimise something that is being measured, rather than focusing on the real organizational goal.

Trends and patterns which can be seen from metrics are much more useful and informative than individual numerical testing metrics values. Thus the collecting and analysing of testing metrics should be a continuous process.

This way trend identification can lead to earlier corrective actions. [31, pp.109-114]

Perhaps the clearest statement about the importance of measurement is Lord Kelvin's: "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind. It may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science."

Testing metrics will be used to answer e.g. to the following questions: What is the size of the software? How long will it take to test the software? How much will it cost to test the software? How many defects have been found during testing? What types of defects have been found? What are the severities of the defects? How many defects have been corrected? How many defects customers have been found? What is the test coverage? How much of the software has to be tested? Will the software be ready on time? How good were the tests? How much did it cost to test the software? Was the test effort adequate? [32, pp.105-126]

In the following chapters there are depicted some of the testing metrics collected and used during and after GGSN testing.

5.2 Testing effort metrics

The distribution of working hours between different testing phases is shown in Figure 11. From the diagram it can be seen that more effort is put on Product Integration in the later product releases. This was a planned change in the testing strategy, because it was realised that Functional Testing was started too early in the first product release (Rel1). During the testing of Rel1 the software was immature when it was handed to Functional Testing. The defects that should have been discovered during Module Testing and Product Integration were found during Functional Testing. In this kind of situation the usage of testing resources was not optimal. Because of the before mentioned change in the testing strategy of Rel2 and Rel3 the effectiveness of testing was increased and the quality of the product was better. But still in future product releases this trend should be continued so that more effort should be put on earlier testing phases. The target for PI should be 20%.

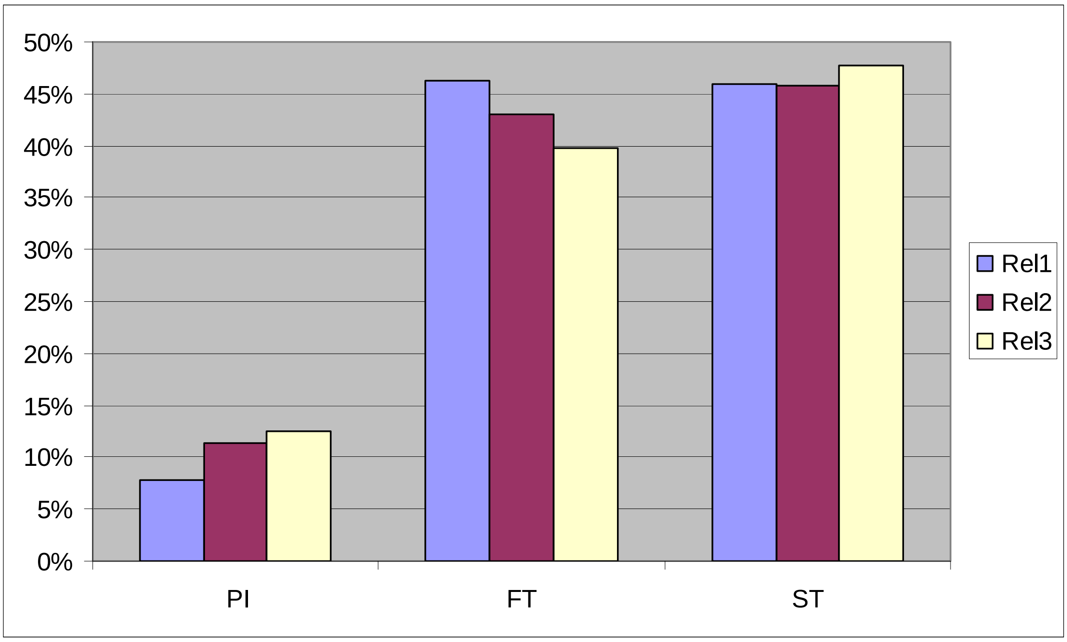

Figure 11 Distribution of working hours by test phases

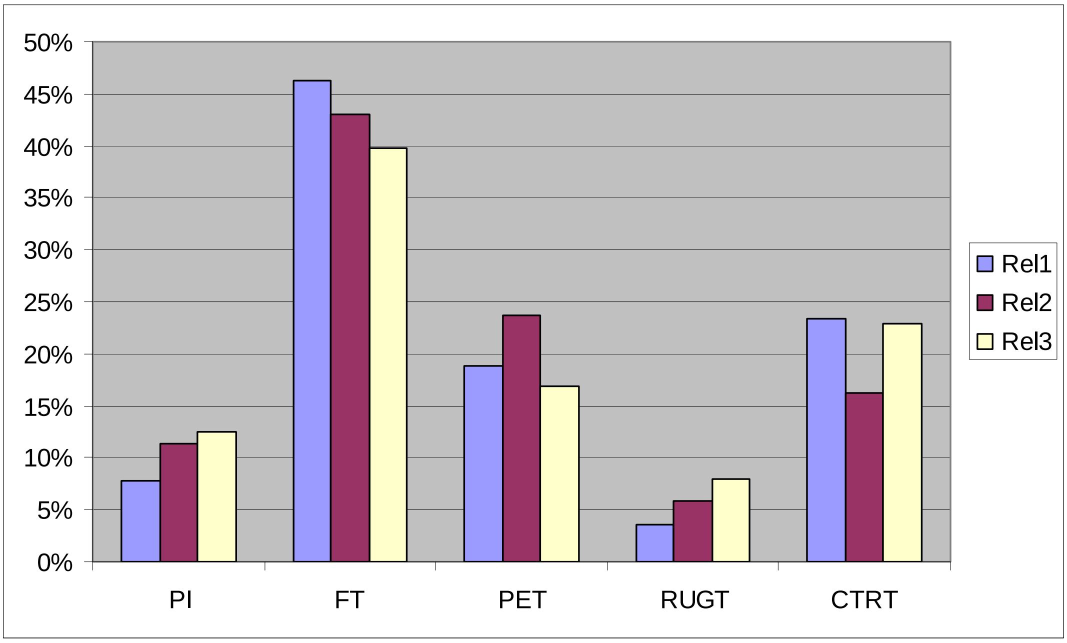

Figure 12 Distribution of working hours by test phases (system testing divided into PET, RUGT and CTRT)

In Figure 12 System Testing (ST) is divided into three testing phases: Performance Testing (PET), Release Upgrade Testing (RUGT), Compatibility and Release Testing (CTRT). Both PET and CTRT testing effort have been about 20% of total testing effort and this seems to be adequate based on the other testing metrics.

5.3 Defect metrics

The number of discovered defects per week is one example of very useful defect metrics shown in Figure 13. This diagram is collected from GGSN Rel2 defects. This diagram shows how many defects have been found in Product Integration, in Functional Testing, in System Testing and by Customers. This kind of diagram facilitates the project milestone decision-making. E.g. at week number 10 the number of defects found in Product Integration testing has decreased to low level compared to weeks 7, 8 and 9. The defect trend gave a strong signal that defect discovery rate in Product Integration testing environment was so low that it was opportune timing to start Functional Testing.

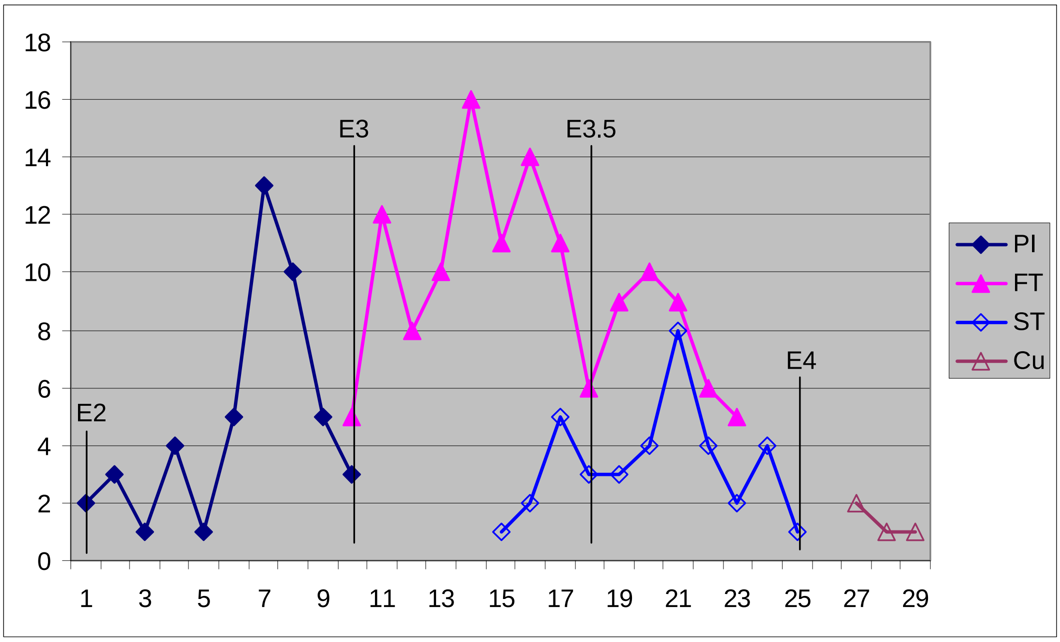

Figure 13 Number of discovered defects per week

In the GGSN product creation process the milestone E3.5 is the official milestone to start System Testing, but there are some defects discovered by System Testing already before E3.5. This is because building up the System Testing environment takes quite much time. Some of the neighbouring network elements used in System Testing are located in other testing laboratories and connections to these neighbouring network elements must be tested before the official System Testing starts. Therefore some pre-tests will be done in System Testing environment to test the connections to neighbouring network elements. This way the official System Testing will start smoothly at E3.5. Quite often already during these pre-tests some compatibility defects will be discovered. It is important to correct these types of compatibility defects as early as possible, because these defects will prevent all more in-depth System Testing.

The milestone decision making is not a trivial task, as from Figure 13 can be seen that the timing of the milestone E3.5 was not optimal. There were quite many defects discovered in Functional Testing after E3.5 and probably the product was not mature enough for System Testing. But the decision had to be done using the information available at week 18. It is easy to be afterwise and say that E3.5 milestone should have been deferred at least two weeks.

During GGSN testing defect metrics information was collected manually from the defect tracking database. This is a time-consuming and error prone task. It is essential that this kind of defect metric diagram is constantly available and up to date. Therefore this diagram should be generated automatically using the information available in the defect tracking database.

Another defect metrics diagram is shown in Figure 14. It presents the percentage of correction not needed defect reports generated by Functional Testing, by System Testing and by Customers. Sometimes testers or users generate unnecessary defect reports because of misunderstanding or misuse. Unnecessary defect reports causes lot of extra work and waste of time and effort in customer care, in testing and in development organisations. Also customers and users become frustrated if the product does not operate predictably and it decreases the customer satisfaction and decreases the experience of product quality. Of course this kind of unnecessary negative experience should be avoided. It can be avoided by improving the customer training, customer documentation and customer support.

The development may waste lot of time when trying to reproduce the unnecessarily reported failure. Quite often failures are hard to reproduce in development environment, because the real network elements are not available. It might require lot of effort before development realises that the reported failure was generated because of misunderstanding or misuse by the user or by the tester. Also the testing department wastes time, because repeating the strange behaviour of the software and testing different configurations when trying to isolate the problem takes lot of time, also collecting the information for the defect report and writing the defect report requires extra effort. The unnecessary defect reports increases also the work load of fault management board, which must analyse and prioritise all defect reports.

Because of all those reasons mentioned above, the number of unnecessary defect reports should be minimized. This can be done by increasing the communication between development, testing and customer care. Also improving the product specification documentation and customer documentation will help. Effective competence transfer from development to other departments is essential.

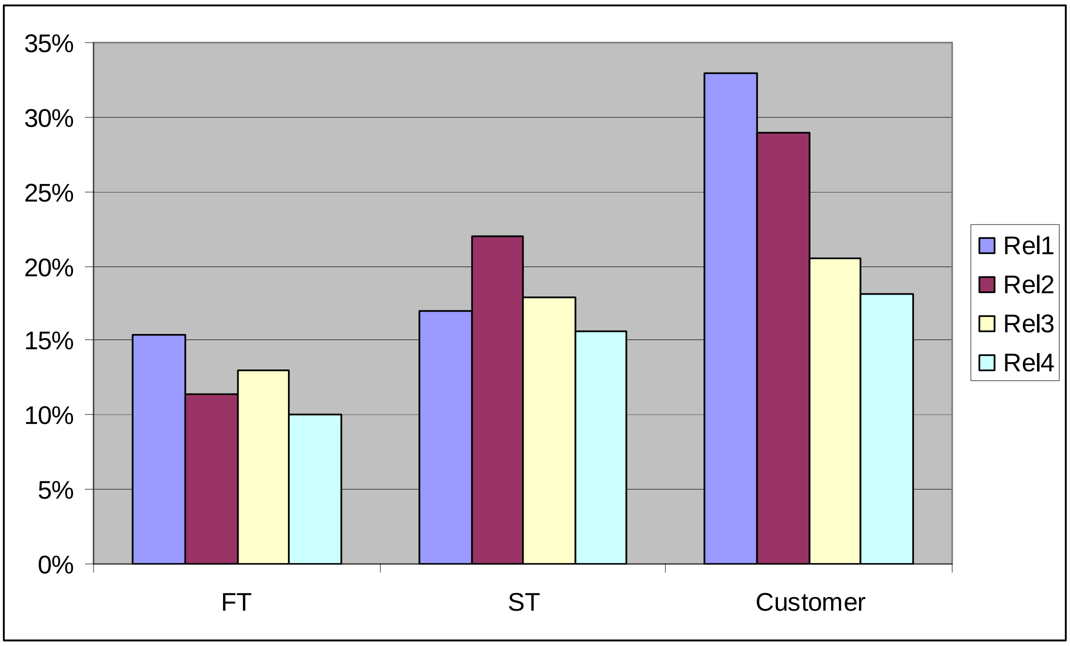

Figure 14 Percentage of “correction not needed” defect reports

As can be seen from Figure 14, the percentage of correction not needed defect reports increases when the distance from development increases. This is logical because communication decreases when distance increases. In all phases there is a declining trend when new product releases are taken into use. This is a good example how the product competence has improved during the evolution of the product in all phases and the product has become more familiar to testers and users. So they reported remarkable lower amount of unnecessary defect reports when using later product releases.

Still for the product release Rel4 the percentages are 10% for FT, 15% for ST and 18% for the customers. The percentages are quite high. A realistic target for the future releases could be 5% lower for each phase. But it must be taken into account that main task for testing is to find errors and report the errors to the development. This defect metric must not be emphasised too much i.e. the threshold to generate a defect report must not increase. The correction not needed defect report metric is a useful tool when estimating the quality of documentation or the training needs to increase the product competence in the testing departments.

5.4 Test case metrics

One example of test case metrics is presented in Figure 15. It shows the percentage of not relevant test cases, i.e. test cases that are planned and the content of the test cases are designed, but during the test execution phase it is realised that these test cases could not be executed. The reason could be e.g.: the requirement has been removed, the feature is not yet implemented, the functionality of the feature is misunderstood during the test planning and the test design phase.

All the time and effort used to plan and design not relevant test cases has been wasted. The methods to prevent the not relevant test cases are quite similar as explained in the paragraph 5.3 e.g.: the requirement specification and the other design specifications must be comprehensive and up to date, the communication between development and testing departments must be smooth, the competence transfer from the development to the testing departments must be efficient.

From Figure 15 it can be seen that the percentage of not relevant test cases in Product Integration and in Functional Testing is about 5% which is reasonable, but for Performance Testing and Release and Compatibility Testing the percentage has been too high (especially for PET). In CTRT there has been made some corrective actions during the evolution of the product, and the percentage of the latest product release is decreased below 5% which is excellent. The target percentage of not relevant test cases for each test phase should be below 5%.

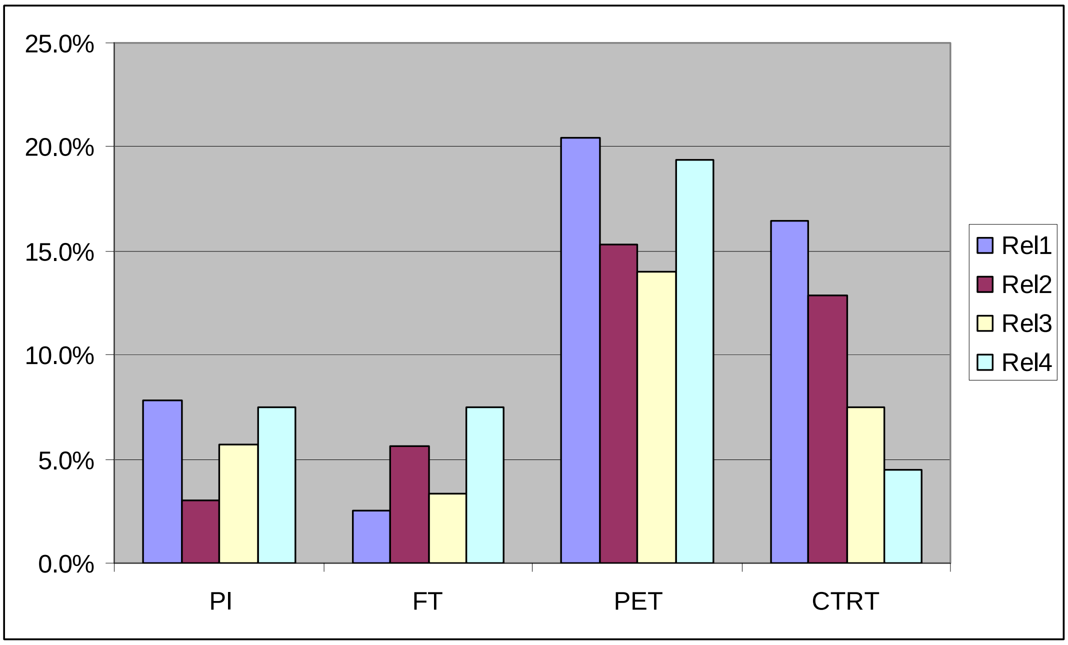

Figure 15 Percentage of not relevant test cases

5.5 Testing effectiveness metrics

Testing effectiveness can be demonstrated e.g. as shown in Figure 16 Hours needed to discover one defect. When calculating the hours then test planning, test design, test execution and test reporting must be taken into account. For confidentiality reasons the values in the diagram are presented proportional to Functional Testing, i.e. CTRT testing requires 3.5 times more hours than FT, but the real amount of hours is not shown in the diagram.

The reason, why CTRT requires remarkable more hours to discover one defect compared to PI and FT, is that the testing environment is more complex in CTRT and it takes more time to configure the real network elements. In PI and in FT simulators are used, therefore the testing is faster. Also the product is more mature when it is taken into CTRT testing and there are less remaining defects to discover.

The reason why it takes two times more time to discover one defect in Performance Testing than in Compatibility and Release Testing is that the main task for CTRT is to detect errors, but PET has also another task, namely PET has to measure the performance figures of the product. Sometimes if the measured performance figures of the product are lower than planned, then it is not a defect, it is a feature.

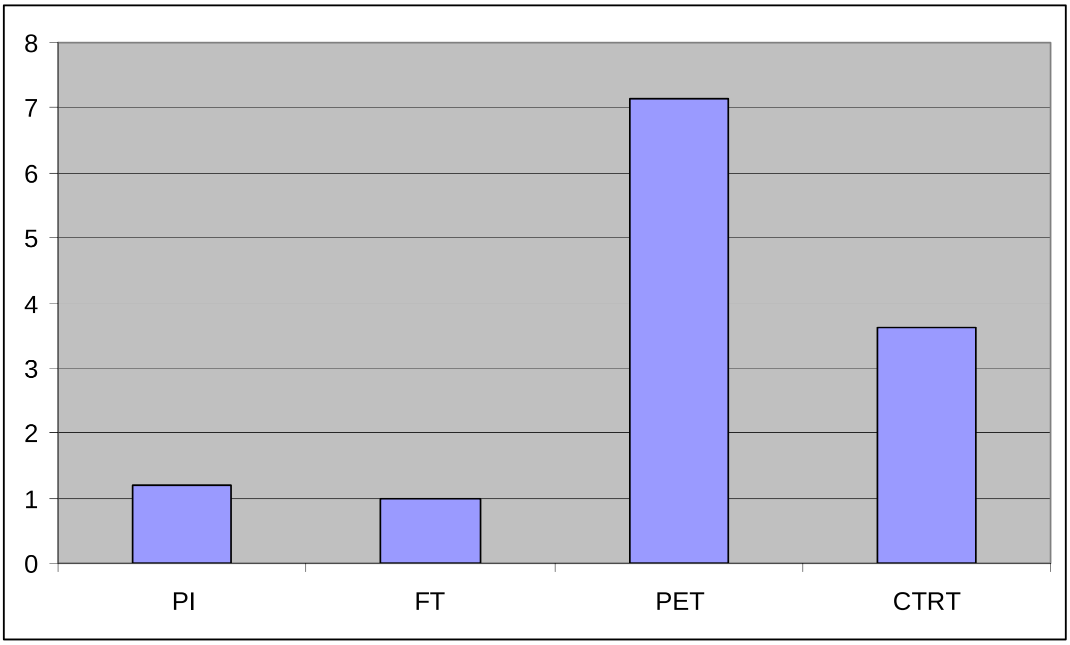

Figure 16 Hours needed to discover one defect

Another example of testing effectiveness metrics is shown in Figure 17. It shows the number of test cases needed to discover one defect in different test phases. Again for confidentiality reasons the proportional presentation is used. In this case other test phases are compared to Product Integration.

The reason, why in PI lesser test cases are needed to discover one defect than in FT, is that the testing environments are quite similar but the product is more immature when it is in PI phase.

The reason for high value for PET is that again PET has also the measurement task in addition to the error detection task.

The explanation for CTRT value is that the test cases are more complex in CTRT than in PI and in FT. In CTRT one test case can cover several interfaces and protocols, whereas in PI and in FT one test case covers e.g. one information element of one protocol of one interface. Therefore the probability to discover a defect by one test case in CTRT phase is higher than in PI and FT phases.

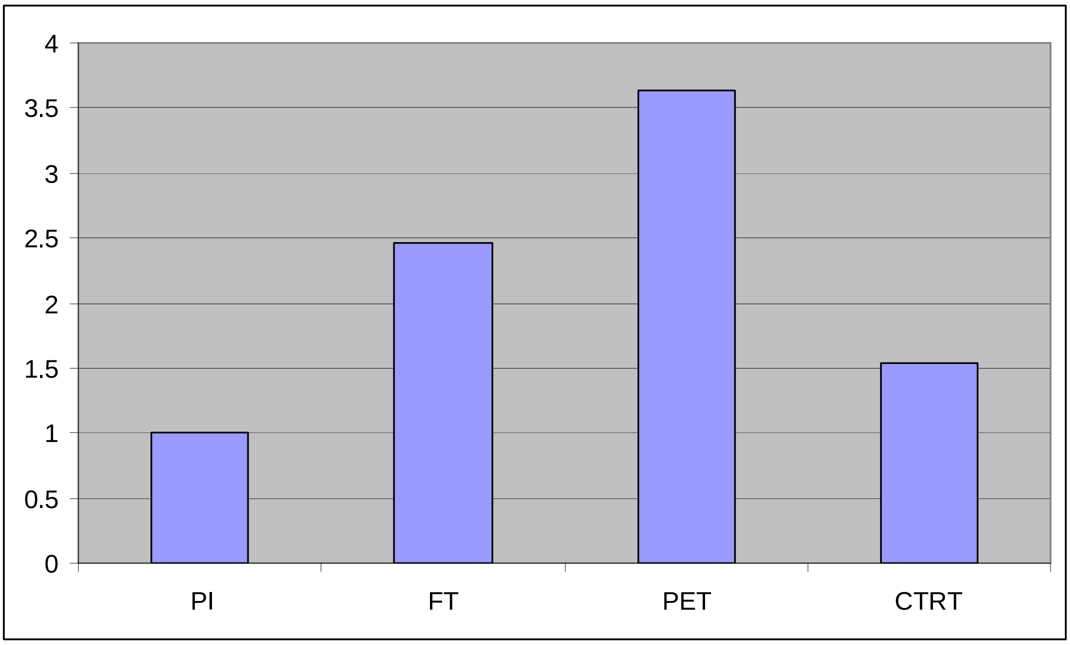

Figure 17 Number of test cases needed to discover one defect

In Figure 16 and in Figure 17 the information is collected only from one release of the product. It would be much more informative if the information has been collected during the whole evolution of the product, then the trends of each test phases can be seen. Based on the changes in the trends some corrective actions can be performed.

5.6 Summary and discussion

There are huge amount of possible testing metrics that can be collected and calculated for each software project. Plenty of raw data is available in the working hour reporting database, in the requirement management database, in the test management database and in the fault management database. But the questions are: how accurate is this data and how easily the raw data can be refined into readable information, preferably into graphic representation. Everyone has his own way of doing things. Someone inputs minimum amount of information into the database. Someone fills out all possible data fields, but unfortunately differently than someone else. If there are not common instructions and practices what data should be saved into the databases, then it is hard to get the needed information out from the database. At least the extracted data must be processed and filtered manually to get the general view of the situation. There are tens of people inputting the data into these databases. Therefore the personnel must be trained and it must be explained why the data has to be saved into the databases and what is the benefit for the project and for each person.

During this study all the testing metric information had to be extracted from the databases manually and the data had to be sorted, filtered and printed into graphic form. This takes too much time and effort. If the purpose is to use the testing metrics during the test process, then e.g. up to date defect metrics must be available on daily basis. Then automated scripts and macros must be used. If testing metrics really want to be used and the target is to get benefit from it, then testing metrics system must be build up and it needs management support and resources. Also resources must be allocated for the maintenance work.

Because the testing metrics are not unambiguous, an error margin of testing metrics is sometimes quite difficult to estimate. When the decisions are based on several testing metrics, then the metrics complement each other. This way the probability for correct decisions will be higher.

Everyone can imagine how difficult it is to fly an aeroplane in a fog without flight instruments (altimeter, compass, air speed, radar, etc.). The same applies when the test manager is trying to control the software testing process without testing metrics. It is as flying eyes closed.

6. DEFECT ANALYSIS

This chapter presents the defect analysis method which was used in this thesis.

The purpose of defect analysis is to find the error prone functionality areas of the product. During this thesis work defects reported by customers were analysed using the matrix presented in Table 5. For confidentiality reasons all information in Table 5 is fictional. It must be analysed what functionality area each defect affects. If a defect is critical, major or minor, then a value 3, 2 or 1 is given correspondingly. After all defects have been analysed, then values in each column are summed up into the column "Total".

Using this kind of defect analysis method, error prone functionality areas of the software can be determined easily. This is valuable information when planning the testing of next software release, because probably those error prone areas of the system are complex and quite often the same areas have errors also in next releases. This is called Pareto principle or 80/20 rule. Barry W. Boehm stated in his book, Software Risk Management [33, p.123]: "Many software phenomena follow a Pareto distribution: 80% of contribution comes from 20% of the contributors". Example: 20% of software modules contribute 80% of the errors. The defect analysis matrix also reveals the features that are used by customers and are important to customers. In this issue again Pareto principle applies.

The purpose of lower part of Table 5 is to examine if there are some common aspects in discovered defects, such as: error handling problems, core dump or required testing method. Also following information is analysed: where a defect is discovered, where the defect should have been discovered and where the defect is injected into the system i.e. root cause.

<u>Error handling:</u> One challenging area of software design and implementation is error handling i.e. situations when something unexpected occurs. In the defect analysis it is advantageous to examine if error handling has been implemented carelessly or testing has been inadequate. Based on the defect analysis feedback can be given to software

development or/and testing of special unexpected situations can be improved in future.

Core dump: In some error situations the kernel process or the application process crashes. These severe situations cause unforeseeable consequences. So it is important to carefully look after this kind of defects. Application or kernel crashes occur normally only in early testing phases when software is still immature. This value gives a rough estimate of software stability and this value is a useful testing metric when entry to higher level testing phases is considered. If the software is unstable it is not worthwhile to start higher level testing.

Required testing method: Sometimes, because of time and resource constraints, testing does not concentrate enough on stability and load testing issues. In the area “Required testing method” in Table 5 is examined if more extensive long term testing or load/overload testing is needed to discover memory leakage or buffer overflow problems. High amount of escaped defects that could be detected by regression test cases illustrates inadequate regression testing and/or e.g. imperfect configuration management in software development.

Defect discovered in: The following area in Table 5 is “Defect discovered in”. In this example only customer defects are analysed, but this defect analysis should be broadened to cover all defects discovered in all testing phases. This way Defect Removal Effectiveness (DRE) of each test phase can be evaluated. In Table 5 also the severity of each defect is taken into account.

Defect Removal Effectiveness can be calculated by equation (1).

$$\mathrm{DRE} = \frac{D}{D + E} \times 100\%, \qquad (1)$$

where D = Defect discovered in the test phase

E = Defects escaped

If summed values for each test phase in Total column are e.g.:
CTRT=197, SyVe=96, Customer piloting=15, Customer actual use=21.
Then DRE of CTRT testing = 197 / (197 + 96 + 15 + 21) = 60%.

Defect should have been found in: This area of Table 5 gives also useful information where more testing effort should be added. It is essential that defects are discovered as early as possible and in “correct” testing environment. The meaning of each testing environment and testing phase is to detect efficiently definite type of defects. Otherwise it is waste of testing equipment resources and testing personnel resources. Testing in a complex environment takes more time and effort than in a simple environment. Hence defects must be discovered in as simple testing environment as possible. This saves time and money.

Root cause: The information about origin of the defect gives necessary feedback to specification, design and implementation phases of a software product. This feedback can be used to make corrective actions to increase effectiveness and efficiency of the left hand side of the software development V-model Figure 9.

In this thesis work the defect analysis has been done manually and it is very time consuming task to collect the required information. But a defect-tracking database has quite many features, which will support this kind of defect analysis. The defect analysis should be a continuous process during the software development and testing. Defect analysis should give numerical and graphical information about defect status on a daily basis. Then development and testing can operate dynamically and concentrate on the most critical and high-risk issues. Defect analysis is also a powerful tool to facilitate project management decision-making. The only manual work should be inputting the defect information into the defect-tracking database. All numerical and graphical defect analysis printouts should be generated automatically by the defect tracking system.

A prerequisite for a functional and valuable defect analysis is valid, accurate and real-time defect data in the defect-tracking database. This needs an independent, objective and competent defect follow-up team.

Also the people using the information generated by the defect analysis should be trained, so they do not draw limited and biased conclusions of the information generated by defect analysis.

Table 5 Defect analysis of defects reported by customer

Problem ID		1-45739391	1-47573867	1-47656701	1-47962580	1-48084751	1-49489655	1-49742718	1-49809995	1-50365061	1-50598721	1-51555748	1-51734791	1-51793004	1-51798221	1-51937081	1-52026486	1-52026488	1-52176483	1-52185791	1-52312650	1-52413083	1-52418022	1-52522321	1-52595064	
Severity		B - Major	A - Critical	C - Minor	C - Minor	C - Minor	A - Critical	B - Major	B - Major	B - Major	C - Minor	B - Major	B - Major	B - Major	C - Minor	B - Major	B - Major	B - Major	C - Minor	B - Major	C - Minor	C - Minor	B - Major	C - Minor	C - Minor	**Total**
Functionality	**Sub area**																									
Charging	Volume based charging						3															1				4
	Time based charging																									0
	Hit based charging																									0
Traffic analysis	Layer 4									2																2
	Layer 7															2										2
Operating and maintenance	NetAct	2																								2
	Graphical user interface															2										2
	Command line interface																									0
	Configuration management																									0
	Fault management	2																								2
	Performance management				1																		2			3
Tunnelling	L2TP																	2								2
	GRE																									0
	IPinIP		3									2														5
User plane traffic	HTTP								2								2									4
	WAP1.X																				1					1
	WAP2.0										1															1
	MMS																									0
	RTSP							2												2						4
Neighbouring network element or protocol	SGSN																									0
	CG												2								1					3
	OSC diameter					1																	2		1	4
	OSC radius																									0
	RADIUS authentication									2																2
	RADIUS accounting																		1							1
	LDAP																									0
	NetAct	2																								2
Error handling	Unexpected message																									0
	Erroneous message content		3																1							4
	Special message timing					1								2								1				4
	Neighbouring network element down																									0
Core dump	Application								2					2												4
	Kernel		3																							3
Required testing method	Long term testing								2																	2
	Load testing				1																					1
	Overload testing						3							2										1		6
	Regression testing			1																						1
Defect discovered in	Product Integration																									0
	Functional Testing																									0
	Performance Testing																									0
	Compatibility and Release Testing																									0
	System Verification																									0
	Customer piloting	2	3	1	1	1	3	2	2	2	1	2	2	2	1	2	2	2								31
	Customer actual use																		1	2	1	1	2	1	1	9
Defect should have been found in	Module Testing	2										2						2	1				2			9
	Product Integration			1			3			2						2					1				1	10
	Functional Testing					1					1			2			2									6
	Performance Testing							2							1											3
	Compatibility and Release Testing		3		1								2							2						8
	System Verification								2													1				3
Root cause	Requirement specification								2						1							1				4
	Implementation specification				1	1	3			2	1			2									2	1	1	14
	Software design	2																	1	2						5
	Software implementation		3					2		2		2	2			2	2	2			1					18
	Software configuration management			1																						1

6.1 Summary and discussion

In this study there were found a couple of critical functionality areas of the product, which will definitely need more weighted testing in the future releases. Also the development and testing should concentrate more on the error handling.

The defect analysis can be done after the product is released as done in this study and then the information created by the defect analysis can be utilised in the next product release test planning. The information will be used in the product risk analysis, which will be described in the following chapter.

Or the defect analysis can be used during the software testing. In this method the defect analysis can be used as a real-time testing metric, which will facilitate the decision-making in the milestone meetings. It is important to analyse all defects discovered in all testing phases.

A real-time defect analysis tool should be built up. It should display in graphical form the following information: number of faults found in each test phase, number of faults found in each functionality area of the product, the response time from development and the fault correction time.

7. PRODUCT RISK ANALYSIS

This chapter describes how the product risk analysis was performed based on the information received from the defect analysis.

“In most cases ‘what’ you test in a system is much more important than ‘how much’ you test” [26, p.25].

“Prioritise tests so that, when ever you stop testing, you have done the best testing in the time available” [34, p.12].

Before test planning, test design and test execution product risk analysis will be done. For confidentiality reasons all information in Table 6 is fictional. As shown in Table 6 functionality, subarea and “Total” column values are copied from defect analysis of earlier product release Table 5. New functionalities and subareas are added to table. After this, requirements are added to columns and each requirement is analysed what functionalities and subareas it is interrelated to. Then value 3 is given to all requirements as default value in the row “Error prone requirement”. Now each requirement is estimated if it has higher or lower probability for error. The value is adjusted to 4 or 5 if the requirement is interrelated to error prone functionalities, or the value is adjusted to 2 or 1 if requirement is not interrelated to error prone functionalities.

After this values to “New feature” row are filled. Again the default value is 3. If the requirement is totally new the value will be changed to 5. If the requirement has been supported already in earlier releases the value will be set to 2 or 1. This estimation can be done using the information in the product requirement specification.

Then complexity of requirements is estimated. The software architect of the product can give the best estimation about this. Again the default value is 3, and if the requirement is complex then the value is set to 4 or 5, if the requirement is straightforward then the value is adjusted to 2 or 1.

Then the likelihood of failure L can be calculated by equation (2).

$$L = (E + N + C) / 3, \qquad (2)$$

where E = error proneness of the requirement (1…5)
N = newness of the requirement (1…5)
C = complexity of the requirement (1…5)

Then “Impact on the user” is estimated. Product marketing and customer support personnel has the best knowledge about customer needs and how the product will be used. Again the default value is 3 and it can be adjusted higher or lower depending on the importance.

Finally “Risk priority” will be calculated as follows:

“Risk priority” = “Likelihood of failure” x “Impact on the user”

Table 6 Defining risk priority for each requirement

Functionality	Sub area	Defect analysis	Requirement 1	Requirement 2	Requirement 3	Requirement 4	Requirement 5	Requirement 6	Requirement 7	Requirement 8	Requirement 9	Requirement 10	Requirement 11	Requirement 12	Requirement 13	Requirement 14	Requirement 15	Requirement 16	Requirement 17	Requirement 18	Requirement 19	Requirement 20
Charging	Volume based charging	4				X		X								X						
	Time based charging	0			X			X			X											
	Hit based charging	0						X							X				X			
Traffic analysis	Layer 4	2			X				X			X	X	X								
	Layer 7	2				X			X						X		X			X		
Operating and maintenance	NetAct	2								X							X					
	Graphical user interface	2		X	X																	
	Command line interface	0				X									X							
	Configuration mgmt	0								X												X
	Fault management	2								X												
	Performance mgmt	3								X						X					X	
Tunnelling	L2TP	2							X								X					
	GRE	0										X										
	IPinIP	5														X				X		
User plane traffic	HTTP	4				X	X							X	X	X						
	WAP1.X	1		X																		
	WAP2.0	1						X			X											
	MMS	0						X														
	RTSP	4				X										X						
	FTP								X													
	PoC										X	X										
	Email									X						X						
Neighbouring network element or protocol	SGSN	0					X					X					X	X				
	CG	3			X	X	X															
	OSC diameter	4						X	X	X												
	OSC radius	0									X											
	RADIUS authentication	2	X		X					X				X	X	X	X					
	RADIUS accounting	1	X							X					X			X	X			
	LDAP	0					X					X	X									
	NetAct	2																		X	X	
	IMS											X										
	LIG							X								X				X		
	CNSM										X										X	X
Error prone requirement			2	2	3	4	4	4	4	4	3	3	1	3	3	5	2	1	1	5	3	3
Requirement already in other network element			-	-	-	-	-	-	-	-	-	-	-	X	-	X	-	X	X	-	-	-
Requirement already in release 1			-	-	-	-	-	-	-	X	-	-	-	-	-	-	X	-	-	-	-	-
Requirement already in release 2			X	-	X	-	X	X	X	X	X	X	X	-	-	-	X	-	-	-	-	X
Requirement is new in release 3			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
New requirement			2	5	2	5	2	2	2	1	2	2	2	3	5	3	1	3	3	5	5	2
Complexity of requirement			2	2	3	5	2	2	2	2	4	2	4	4	5	4	4	4	2	2	2	4
Likelihood of failure			2	3	3	5	3	3	3	2	3	2	2	3	4	4	2	3	2	4	3	3
Impact on the user			5	2	5	5	5	5	5	5	2	2	5	5	5	5	5	4	1	5	5	4
RISK PRIORITY			**10**	**6**	**13**	**23**	**13**	**13**	**13**	**12**	**6**	**5**	**12**	**17**	**22**	**20**	**12**	**11**	**2**	**20**	**17**	**12**

In Table 7 the requirements are sorted according to the risk priority. In Table 8 the requirements are divided into three groups: red = high-risk requirements, yellow = medium risk requirements and grey = low risk requirements.

Table 7 Sorting requirements according the risk priority

	Likelihood of failure	Impact on the user	RISK PRIORITY
Requirement 4	4.7	5	23
Requirement 13	4.3	5	22
Requirement 14	4.0	5	20
Requirement 18	4.0	5	20
Requirement 12	3.3	5	17
Requirement 19	3.3	5	17
Requirement 3	2.7	5	13
Requirement 5	2.7	5	13
Requirement 6	2.7	5	13
Requirement 7	2.7	5	13
Requirement 20	3.0	4	12
Requirement 8	2.3	5	12
Requirement 11	2.3	5	12
Requirement 15	2.3	5	12
Requirement 16	2.7	4	11
Requirement 1	2.0	5	10
Requirement 2	3.0	2	6
Requirement 9	3.0	2	6
Requirement 10	2.3	2	5
Requirement 17	2.0	1	2

Table 8 Dividing requirements into three groups: high, medium and low risk requirements

	Likelihood of failure	Impact on the user	RISK PRIORITY
Requirement 4	4.7	5	23
Requirement 13	4.3	5	22
Requirement 14	4.0	5	20
Requirement 18	4.0	5	20
Requirement 12	3.3	5	17
Requirement 19	3.3	5	17
Requirement 3	2.7	5	13
Requirement 5	2.7	5	13
Requirement 6	2.7	5	13
Requirement 7	2.7	5	13
Requirement 20	3.0	4	12
Requirement 8	2.3	5	12
Requirement 11	2.3	5	12
Requirement 15	2.3	5	12
Requirement 16	2.7	4	11
Requirement 1	2.0	5	10
Requirement 2	3.0	2	6
Requirement 9	3.0	2	6
Requirement 10	2.3	2	5
Requirement 17	2.0	1	2

7.1 Summary and discussion

In this chapter the product requirements are analysed. The objective is to find the most critical requirements i.e. the requirements that have high probability of errors and the requirements that are most important to the customers.

Some improvements into the requirement specifications of the future product releases should be made. For each requirement the functionality subarea that the requirement is related to should be documented in the requirement specification. This will facilitate the search of the error prone requirements, because in the defect analysis the error prone functionality subareas are found.

In addition an estimation of the complexity of each requirement and information about the importance of each requirement should be added to the requirement specification. It is essential that the requirement specification will be updated regularly, because the estimation of the complexity of the requirement might vary during the development process and also sometimes customers’ opinions may change or new information from new customers could be received. The product marketing and the customer support are the main information sources of the customers’ needs.

In the next chapter the test planning will be done based on the risk analysis.

8. TEST PLANNING, DESIGN AND EXECUTION

This chapter shows how the test planning can be done based on the product risk analysis and how the needed test design and test execution effort can be reduced.

By studying the content of each requirement it is planned how all functionalities of the requirement can be verified. This may need several test cases or the requirement might be possible to verify thoroughly by one test case. In the test planning phase only titles of test cases are defined. When test planning is finished the number of test cases for each requirement is known.

Test design i.e. designing content for each test case is the following task. It is not necessary to design content for all test cases, because the information created by product risk analysis can be used to drop some test cases of medium and low risk requirements. The test case selection in Table 9 can be done as follows: all test cases for high risk requirements will be designed and executed, about 70% of test cases of medium risk requirements will be designed and executed, about 30% of test cases of low risk requirements will be designed and executed.

This kind of risk based testing reduced the needed test design and test execution effort in this example Table 9 by 30% without jeopardising the quality of the product.

The order in which the test cases are executed is also crucial, because it is important that the high-risk requirements are verified in early phase. If defects will be found in these requirements then software development will have enough time to correct them and also thorough retesting can be done. In some projects there might happen unexpected changes in schedule and the time frame for testing will be reduced and the product must be delivered to e.g. customer trial earlier than planned. It is important that at least the high-risk requirements are verified and at least the existing problems in high-risk requirements are known and customers can be informed.

Table 9 Defining what test cases will be designed and executed

Requirement and test case ID	RISK PRIORITY	Test case will be designed	Test case will be executed
Requirement 4	23		
Test case 4.1		Yes	Yes
Test case 4.2		Yes	Yes
Test case 4.3		Yes	Yes
Requirement 13	22		
Test case 13.1		Yes	Yes
Requirement 14	20		
Test case 14.1		Yes	Yes
Test case 14.2		Yes	Yes
Test case 14.3		Yes	Yes
Test case 14.4		Yes	Yes
Test case 14.5		Yes	Yes
Requirement 18	20		
Test case 18.1		Yes	Yes
Test case 18.2		Yes	Yes
Requirement 12	17		
Test case 12.1		Yes	Yes
Test case 12.2		Yes	Yes
Test case 12.3		Yes	Yes
Requirement 19	17		
Test case 19.1		Yes	Yes
Test case 19.2		Yes	Yes
Test case 19.3		Yes	Yes
Test case 19.4		Yes	Yes
Requirement 3	13		
Test case 3.1		Yes	Yes
Test case 3.2		Yes	Yes
Test case 3.3		Yes	Yes
Test case 3.4		Yes	Yes
Test case 3.5		No	No
Test case 3.6		No	No
Requirement 5	13		
Test case 5.1		Yes	Yes
Test case 5.2		Yes	Yes
Test case 5.3		Yes	Yes
Test case 5.4		No	No
Requirement 6	13		
Test case 6.1		Yes	Yes
Test case 6.2		Yes	Yes
Requirement 7	13		
Test case 7.1		Yes	Yes
Requirement 20	12		
Test case 20.1		Yes	Yes
Test case 20.2		Yes	Yes
Test case 20.3		No	No
Requirement 8	12		
Test case 8.1		Yes	Yes
Test case 8.2		Yes	Yes
Test case 8.3		No	No
Requirement 11	12		
Test case 11.1		Yes	Yes
Test case 11.2		Yes	Yes
Test case 11.3		Yes	Yes
Test case 11.4		Yes	Yes
Test case 11.5		No	No
Test case 11.6		No	No
Requirement 15	12		
Test case 15.1		Yes	Yes
Test case 15.2		Yes	Yes
Test case 15.3		No	No
Requirement 16	11		
Test case 16.1		Yes	Yes
Test case 16.2		No	No
Test case 16.3		No	No
Requirement 1	10		
Test case 1.1		Yes	Yes
Test case 1.2		No	No
Requirement 2	6		
Test case 2.1		Yes	Yes
Test case 2.2		No	No
Test case 2.3		No	No
Requirement 9	6		
Test case 9.1		Yes	Yes
Test case 9.2		Yes	Yes
Test case 9.3		No	No
Test case 9.4		No	No
Test case 9.5		No	No
Requirement 10	5		
Test case 10.1		Yes	Yes
Test case 10.2		No	No
Requirement 17	2		
Test case 17.1		Yes	Yes
Test case 17.2		No	No
Test case 17.3		No	No

All test cases will be designed and executed

About 70% of test cases will be designed and executed

About 30% of test cases will be designed and executed

The test case list Table 9 will be defined during the test planning phase. At that time the testing of the new product release has not been started in any test phases. The test case list is planned based on the information received from the defect analysis of the previous product releases and on the information received from the requirement analysis. But during the test execution phase it is very important to reassess the requirement and test case list regularly, because the real-time defect analysis of the new product release might give some new information about the error prone areas. Of course the focus of the testing should be changed accordingly.

“Prioritise tests so that, when ever you stop testing, you have done the best testing in the time available” [34, p.12].

8.1 Summary and discussion

This chapter explained how the product risk analysis can be used as a foundation for test planning, test design and test execution. With this method the required effort and time can be reduced remarkably and still a high quality product will be delivered to the customers.

The same functionality subarea division, which was used in the defect analysis and in the risk analysis, should be used also in the test cases. For each test case it should be defined what functionality subareas the test case is related to. This will help when estimating the test coverage and the focus of the testing. The usage of the test management database should be developed so that the test coverage metrics can be generated automatically and in real-time. It will help to adjust the testing dynamically.

The highest level test document is the Master Test Plan. It is the project plan that covers all testing phases. The testing strategy must be defined in the Master Test Plan. In the testing strategy the guidelines of the testing should be defined. The testing strategy should define what requirements and how thoroughly each of them will be tested in the different testing phases. This way the focus of the testing in each testing phase can be directed to the correct requirements and unnecessary overlapping work can be avoided.

The importance of the test planning can not be emphasized too much. The old carpenter’s adage “Measure twice, cut once” applies also very well in the software testing.

9. SUMMARY AND CONCLUSIONS

This chapter summarises the work done in this thesis. It evaluates the research method and the results of the study. Conclusions of the thesis are presented. Finally, future work how the test process could be developed is discussed.

9.1 Summary

The software testing costs are about 40% of the total software product development costs, and an exhaustive testing takes impractical amount of time. Therefore it is very important to plan the software testing carefully in order to keep the testing costs, schedule, and product quality in the acceptable level. The test planning and the testing is a challenging task. It requires tradeoffs between cost, time and quality. In this thesis it is assumed that the risk based testing is the solution by which the high testing efficiency and the high product quality can be achieved. The target of the risk based testing is that the testing will be focused on the error prone features and on the features that are the most critical to the customers. The necessary information needed in the risk based testing can be collected by the testing metrics, the defect analysis, the requirement analysis and the product risk analysis.

In this thesis the functionality of the packet core network and the GGSN is explained in Chapters 2 and 3. Software testing in general is explained in Chapter 4. The practical part of the thesis started with the testing metrics in Chapter 5. During the test process analysis it was realised that in the GGSN testing the testing metrics has not been collected and used systematically. In Chapter 6 the post-test defect analysis of customer defects was performed. It revealed some error prone functionality areas of GGSN. The weakness of the defect analysis was that the defects discovered by the in-house test phases was not analysed, because of a lack of time. It would have given much more extensive picture of the product. The product risk analysis method was introduced in Chapter 7. Based on the product risk analysis and requirement analysis the test planning of the upcoming product release was done. This kind of test planning will reduce the needed test design and test execution effort remarkably. The real effect on the product quality can not be estimated yet, because the product is still in the testing phase and the new software release is not yet delivered to customers.

9.2 Evaluation of research method

During this thesis work both theoretical and empirical approach was used to analyse the testing process. Participating over two years in the practical testing work in the GGSN CTRT testing laboratory was the kick-off for this thesis. Based on the practical work experience the software testing literature was studied and the ideas/problems presented in the literature were compared with the ideas/problems faced in the real testing work. This gave broad understanding of the situation. The literature study confirmed the idea that the risk based testing could be the solution for the GGSN testing. The testing metrics also confirmed some problems faced in the real testing work.

There was lot of raw data available for the defect analysis and for the product risk analysis in the databases. Lot of data mining was needed to collect the necessary information. This took more time than estimated. Therefore, the analysis was done mainly from the CTRT testing point of view, and only the defects reported by customers were analysed. After the product risk analysis was done the test planning was quite straightforward. Hopefully the ideas presented in this thesis will be taken into use in the testing of the coming product releases. Then the testing metrics will probably show increased testing effectiveness and better product quality.

9.3 Conclusions

The testing metrics is a powerful tool for test management, but collecting the information manually takes too much time and effort. Also the raw data in the working hour reporting database, in the defect tracking database and in the test case database is not uniform. Hence the personnel, who input the raw data into the databases, should be trained and common practices should be taken into use in order to get the databases in such state that testing metrics can be generated automatically. The testing metrics should be available in real time. In that case the testing process can be adjusted dynamically during the testing.

The defect analysis gives valuable information about the quality of the software and also information about customers’ needs. The defect analysis reveals the error prone features in which the testing should concentrate. The first phase of the defect analysis is that all the customer defects are analysed as was done in this thesis. During the testing of the future GGSN releases also all the defects reported by all different testing phases should be analysed.

An intensive co-operation between the product development, all different testing phases, product management, marketing, customer support and the customers is needed when the product risk analysis is done. This kind of extensive co-operation will produce comprehensive knowledge about the product risks. Based on the known risks the available testing time and the available testing resources can be used as efficiently as possible and the customers will get the high quality product on time.

Early involvement of the testing personnel will increase the communication and co-operation between the development personnel and the testing personnel. This will increase the testing efficiency, because less “not relevant test cases” will be planned and designed, and less “correction not needed defect reports” will be generated. At the moment the competence of GGSN testing teams is at such level that also error prevention methods can be used by the testing teams i.e. the testing personnel should participate more into inspections and reviews in the specification and design phases of the future product releases.

9.4 Future work

This thesis pointed out some areas in the testing process which could be changed and improved. By analysing the testing process it is quite easy to find the deficiencies in the testing process but the difficult part of the work is to implement the changes in the testing tools and practises, and to motivate the personnel to change their working methods. This needs strong management commitment, resources and training.

The most important future work is to harmonize the usage of requirement management database, test case database and defect tracking database. For each requirement, test case and defect there must be defined the functionality area/areas of GGSN where each requirement, test case and defect is related to. It must be possible to extract statistical information out of these databases in numerical and also in graphical form. These printouts will facilitate the making of the product risk analysis. Also the steering of the product testing will be exact and fast. The release decision making will be easier, because there will be more real-time information available.

Even though the real network elements are used in the compatibility and release testing environment, the testing environment can never fully simulate the real-life situation where GGSN is operating in customer network. Therefore some of the defects will not be possible to discover cost-effectively in the testing environment. One solution to overcome this problem is the close co-operation with customers. With some of the customers it could be agreed that the customer will participate into the GGSN testing before the software is commercially released i.e. the beta testing for GGSN could be taken into use.

The concept of GGSN testing should be broadened so that the testing will start already in the product specification phase by reviews and inspections, and the testing will continue in live networks until the end of the life-cycle of the network element. The testing personnel could be a gateway for the information flow and competence transfer from the development to the customer support and customers and vice versa.

The risk based testing can be summarised as follows:

No risk ==> No test

10. REFERENCES

[1] Hellgren V. GGSN 4.1 Architecture Specification, internal document, Nokia Networks, July 2003, 94 p.

[2] SG40048.00, Nokia 2G SGSN release SG4 Product Description, Issue 4-0, internal document, Nokia Networks, 2004, 79 p.

[3] Nokia 3G SGSN release 2 Product Description, Issue 1.2, internal document, Nokia Networks, 2003, 139 p.

[4] Nokia Border Gateway release 2, product description (version 1.0), internal document, Nokia Networks, 2003, 23 p.

[5] Nokia Lawful Interception Gateway release 3, product description (version 1.0), internal document, Nokia Networks, 2002, 47 p.

[6] Nokia Intelligent Content Delivery release 2, product description, internal document, Nokia Networks, 2004, 76 p.

[7] Nokia Charging Gateway 4.1 Product Documentation, internal document, Nokia Networks, 2004, 109 p.

[8] Hellgren V. Nokia GGN4.2 Architecture specification, version 1.0, internal document, Nokia Networks, August 2004, 156 p.

[9] 3GPP TS 29.060, 3rd Generation Partnership Project, “Technical Specification Group Core Network; General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface (Release 4)” V4.9.0, 3GPP, September 2003, 89 p. http://www.3gpp.org/ftp/Specs/archive/29_series/29.060/29060-490.zip

[10] 3GPP TS 24.008, 3rd Generation Partnership Project, “Technical Specification Group Core Network; Mobile radio interface Layer 3 specification; Core network protocols; Stage 3 (Release 4)” V4.12.0, 3GPP, September 2003, 459 p. http://www.3gpp.org/ftp/Specs/archive/24_series/24.008/24008-4c0.zip

[11] RFC 1701, Generic Routing Encapsulation (GRE), S. Hanks et al., IETF, October 1994, 8 p. http://www.ietf.org/rfc/rfc1701.txt

[12] RFC 2003, IP Encapsulation within IP, C. Perkins, IETF, October 1996, 14 p. http://www.ietf.org/rfc/rfc2003.txt

[13] RFC 2661, Layer Two Tunnelling Protocol "L2TP", W. Townsley et al., IETF, August 1999, 80 p. http://www.ietf.org/rfc/rfc2661.txt

[14] Routing and Tunnelling in Nokia GGSN, Release 5.0, user manual, issue 7, internal document, Nokia Networks, 2004, 42 p.

[15] RFC 2865, Remote Authentication Dial In User Service (RADIUS), C. Rigney et al., IETF, June 2000, 76 p. http://www.ietf.org/rfc/rfc2865.txt

[16] 3GPP TS 29.061, 3rd Generation Partnership Project “Technical Specification Group Core Network; Interworking between the Public Land Mobile Network (PLMN) supporting packet based services and Packet Data Networks (PDN) (Release 4)” V4.8.0, 3GPP, June 2003, 63 p. http://www.3gpp.org/ftp/Specs/archive/29_series/29.061/29061-480.zip

[17] GPRS Network elements overview, training material, internal document, Nokia Networks, 2003, 144 p.

[18] RFC 2131, Dynamic Host Configuration Protocol, R. Droms, IETF, March 1997, 45 p. http://www.ietf.org/rfc/rfc2131.txt

[19] 3GPP TS 32.215, 3rd Generation Partnership Project, “Technical Specification Group Services and System Aspects; Telecommunication management; Charging management; Charging data description for the Packet Switched (PS) domain (Release 4)” V4.5.0, 3GPP, September 2003, 65 p. http://www.3gpp.org/ftp/Specs/archive/32_series/32.215/32215-450.zip

[20] RFC 3377, Lightweight Directory Access Protocol (v3): Technical Specification, J. Hodges, R. Morgan, IETF, September 2002, 6 p. http://www.ietf.org/rfc/rfc3377.txt

[21] Bernabeu J. Radius Interface Specification for the GGSN 4.0 and OSC 1.0, version 1.4, internal document, Nokia Networks, February 2003, 22 p.

[22] Ala-Kujala P. Network Management Interface Description for Nokia GGSN Release 4.1, version 3.0.3, internal document, Nokia Networks, August 2004, 63 p.

[23] Nyberg K. GGN4.1 ISD Interface Specification, version 1.0.3, internal document, Nokia Networks, February 2004, 37 p.

[24] Myers G. The art of software testing, USA, John Wiley & Sons, ISBN 0-471-04328-1, 1979, 177 p.

[25] Hetzel B. Complete Guide to Software Testing, USA, John Wiley & Sons, ISBN 0-471-56567-9, 1988, 284 p.

[26] Craig R., Jaskiel S. Systematic Software Testing, USA, Artech House Publishers, ISBN 1-58053-508-9, 2002, 536 p.

[27] Dijkstra E. W. Structured programming. Proceedings of the Second NATO Conference on Software Engineering Techniques, Rome Italy, NATO, 1969, pp. 84-88.

[28] Dustin E. Effective Software Testing, USA, Addison-Wesley, ISBN 0-201-79429-2, 2003, 271 p.

[29] Bender R. Requirements Based Testing presentation. Proceedings of StarEast 2005, Software testing analysis & review conference, USA Orlando, Bender RBT Inc. 2005, 232 p.

[30] Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems, USA, John Wiley & Sons, ISBN 0-471-12094-4, 1995, 294 p.

[31] Koomen T., Pol M. Test Process Improvement: A practical step-by-step guide to structured testing, Great Britain, Addison-Wesley, ISBN 0-201-59624-5, 1999, 218 p.

[32] Hutcheson M. Software Testing Fundamentals: Methods and Metrics, USA, John Wiley & Sons, ISBN 0-471-43020-X, 2003, 408 p.

[33] Boehm B. W. Software Risk Management, USA, IEEE Press, ISBN 0-8186-8906-4, 1989, 495 p.

[34] Grove Consultants, ISEB testing foundation course material, Great Britain, Grove Consultants, 2003, 98 p.