

Developing Carrier- and Enterprise- Grade Software

Verax Systems Process Overview



Table of contents

Abstract	3
1. Introduction	3
2. The key elements to success	4
3. Project management	5
4. Design standards	6
5. Programming standards	7
6. Standardized development tools and virtualization	7
7. Source code management, build process and continuous integration	8
8. Test tools and environment	9
9. Metrics, KPIs and continual improvement	10
10. Human factor	11
11. Summary	11

Abstract

This paper presents a software development process utilized by Verax Systems in order to facilitate the design and implementation of scalable, highly-available and robust software for cutting-edge telecommunications, financial and enterprise products. The process presented in this paper is applied for both internal product R&D as well as **bespoke software development** engagements.

The prime objective of our approach is to achieve **software robustness** defined as having high availability, reliability and scalability factors.

As a software vendor of software for demanding telecommunications, financial and large enterprise markets, we pay close attention to the quality and robustness of our systems. The set of techniques and procedures described in the documents gives Verax Systems a competitive advantage in today's marketplace.

Intended audience

This paper is a publication created by Verax Systems' experts and specialists. Its purpose is to help and assist our customers and partners in obtaining in-depth information about internal processes and to highlight the most important issues related to carrier- and enterprise-grade software development.

1. Introduction

Carrier-grade is a telecommunications-originated term indicating products that require up to 5 or 6 nines (or 99.999 to 99.9999 percent) time of correct operation. This value translates into 30 seconds (6 nines) or 5 minutes (5 nines) of downtime per year. Carrier-grade software is typically deployed in communications devices (such as switches, routers, GSM base stations or industrial hardware) or mission critical enterprise systems (such as prepaid billing and charging systems, e-commerce, Internet banking, on-line transaction systems and similar). The "carrier-grade" term is used in this paper in a broader context to indicate any kind of software meeting the carrier-grade requirements described below.

A system (consisting of software and hardware) is usually considered to be carrier-grade when the following conditions are met:

- **Availability** is below 5 minutes downtime (software) or 30 seconds (hardware) per year.
- **Reliability** is below 1 failure per 10,000 sessions (four nines, or 99.99%). Reliability can be deemed as the overall quality of the system.
- The system's **scalability** is such that it can serve more than 100,000 concurrent clients.

For some applications, less stringent requirements may be acceptable. For example, a medical equipment controller needs to be reliable, but it does not have to be scalable as it only serves a fixed number of devices.

Some manufacturers also use the term capacity to describe the performance of their systems, which is defined as a throughput achievable on a single processor. Thus, a system designer cannot focus on scalability alone without paying attention to per-processor capacity.

This paper covers a broad range of carrier-grade software development techniques utilized by Verax Systems. We have successfully implemented these techniques in a variety of projects including:

- Telecommunications operations support systems (OSS, <http://www.veraxsystems.com/en/products/ossbss>).
- Embedded WiFi and WiMax hotspot gateway controllers (<http://www.veraxsystems.com/en/products/SPARK>).
- Internet and smartphone banking (<http://www.veraxsystems.com/en/products/ebanking>).
- Monitoring & management of networks, data centers and applications (<http://www.veraxsystems.com/en/products/nms>).
- Other, bespoke systems.

The software development methods outlined in this document have allowed our applications and systems to offer substantially better performance, throughput, scalability and quality than applications developed based on traditional, ad hoc approaches.

Building a carrier-grade system is challenging and costly – the processes and procedures implemented by Verax Systems have helped us to maintain **consistent performance and quality of deliverables** on a wide variety of customer engagements.

2. The key elements to success

At Verax Systems we have identified the following elements as key to the successful development of carrier-grade software (note: these factors are independent of the programming language used, technology or application type):

1. **A formal, systematic approach to project management.** The Verax methodology is based on proven techniques and processes that introduce rigorous standards with any size of engagement to ensure on-time, on-budget, and on-quality performance (see section 3 for details).
2. Enforcing **design standards** based on a proven set of design patterns. Design patterns make the code easier to understand and maintain, however not all of the commonly used design patterns are suitable for carrier-grade programming. It can be a costly experience to determine which designs work and which ones do not (see section 4 for details).
3. Enforcing **programming standards** based on a proven set of coding guidelines and static code analysis. Implementation patterns help to produce code that is readable and understandable by programmers, as they provide guidelines on how the code should be documented. The adoption of programming standards enables Verax Systems to reuse the code in different projects and even further shorten the development process as well as reduce the total cost of software development (see section 5 for details).
4. **A standard set of software development tools and virtualization of environments.** This set defines which tools (including version numbers) are allowed for software development in order to ensure consistent builds across many workstations – compiling same source code for the same target platform must yield identical results. Verax has selected Eclipse as a standard Integrated Development Environment (IDE) for a multitude of programming languages ranging from Java and C++ to web services XML (see section 6 for details).
5. **Source code management, build process and continuous integration.** At Verax Systems we have invested significantly into building a robust seamless environment for dealing with source code in order to achieve:
 - Repeatable build results across all the software development workstations.
 - Ensuring proper versions of tools are used as even slight differences in compiler versions on different workstations can lead to time-consuming problem tracking.
 - Support for multiple target and compilation platforms.
 - Automatic running of test drivers.
 - Calculating code metrics.
 - Automatic checking of accordance of the produced code against the programming standards.

For more information please refer to section 7.

6. **Test tools and environment.** A robust, cross-platform and automated test environment is a key to successful Quality Assurance activities. There are many commercial and open source testing tools and frameworks available. The majority of them support unit testing, however writing a test is still a manual process. It should be noted that testing is a *corrective measure* while a good design and standards enforcement are *preventive*. Please refer to section 8 for detailed information on how Verax Systems tests its software.

- 7. **Metrics, KPIs and continual improvement.** At Verax Systems we strongly believe that measuring the software development process is an absolute must to ensure high quality of final deliverables. Please refer to section 9 for detailed information on how Verax Systems measures its software development process.

Please note, that the items listed above correspond to project stages as defined by many software development models and share one key characteristic: errors made in an earlier stage escalate on the next one, e.g. the best implementation will not help if the design is wrong.

The ways in which Verax Systems addresses each of these issues is described in the subsequent sections of this paper.

3. Project management

In order to ensure that our projects are delivered on time, on budget and to the defined functionality, Verax Systems has developed and standardized on a formal project management (PM) framework. The framework defines policies, procedures, document templates and tools used by Verax project managers and engineers in successfully completing customer engagements. Based on a combination of industry project management standards: (Rational Unified Process – **RUP, PRINCE2**) and **agile software development** methods, the framework scales from simple projects to very complex ones covering all the critical PM areas such as requirements, change, risk, quality and others. The objective of the framework is to provide a **repeatable, proven, uniform and consistent** project delivery across all the engagements.

All Verax Systems projects are executed under control of Project Portfolio Management (PPM) package (<http://www.veraxsystems.com/en/products/apinikb>). PPM is not only an internal tool, but is also visible for the clients (hosted by Verax Systems). Such an approach allows us to improve project communication and provide customers with a better handle on the project. It is also a central point for a project documentation library and a knowledgebase.

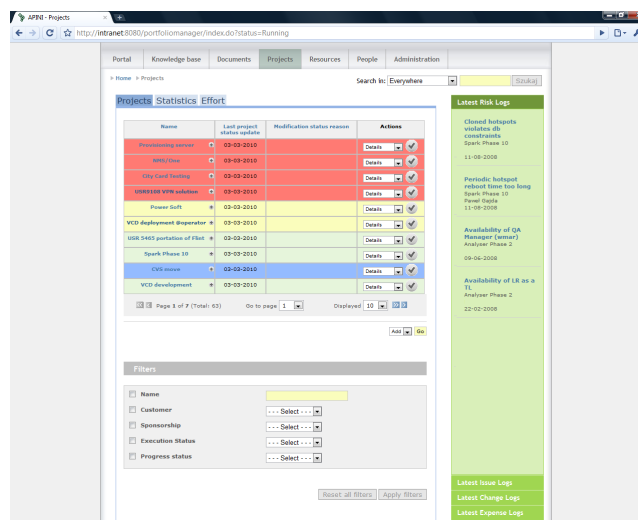


Figure 1: Verax Project Portfolio Management software.

4. Design standards

Design standards at Verax Systems are defined as a series of knowledgebase articles related to GUI design, coding, creation of installation scripts and wizards, build and makefile preparation and others. An important part of the design guidelines is a set of acceptable software design patterns. In general, design patterns for carrier-grade software should be as much reactive, event driven, real-time, and fault tolerant as possible. These requirements exclude many popular and commonly used patterns such as:

- Synchronous events (i.e. RMI or CORBA calls).
- Creating short-lived tasks (processes or threads) to perform work.
- Using non-pooled threads.
- Frequent reading from and writing to a disk or compact flash storage.

For example, the proactor pattern that greatly simplifies asynchronous server code usually cannot be used because it requires too many threads. The event-driven reactor pattern and event de-multiplexing are a much better choice for a system that must respond in real- or sub-real time. It is possible that some patterns applicable in a particular project may not be the best choice in another one and vice versa. Moreover, the application of some patterns may exclude others.

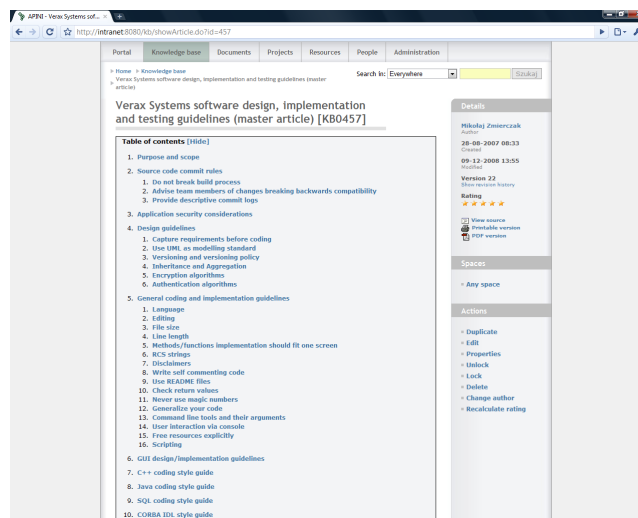


Figure 2: Software design guidelines in the Verax Systems knowledgebase.

The use of design patterns implies object-orientation (OO). OO in carrier-grade system design is desirable because it allows protocols and state machines to take full advantage of polymorphism and inheritance. However, object-oriented design does not necessarily mean that implementation requires an object-oriented programming language. In fact, it is possible to implement an object-oriented design even in assembly language. Verax Systems has implemented its real-time, embedded part of the Cavera platform (<http://www.veraxsystems.com/en/products/cavera>) entirely in C, even though the majority of its functionality is contained in objects.

Another advantage of OO is the ability to use OO design tools such as IBM Rational Rose and a possible future transition to a Model Driven Development approach when mature tools are available.

Enforcing the design standards is a challenge at any software company pressurized by deadlines. Verax Systems enforces the pattern usage on two levels:

- During code reviews performed weekly at project status meetings.
- The design document templates in the document library have sections on design patterns and approaches used, that have to be filled in during the design process.

5. Programming standards

The programming standards defined internally by Verax Systems fall into two categories: **implementation patterns** (general) and **coding style guidelines** (programming language specific). The implementation patterns describe the standard structure for functions, classes and modules (e.g. function tracing, argument validation, etc.). The coding style guidelines refer to how code is laid out and formatted (e.g. variable naming conventions, commenting, etc.). The benefits of these standards are:

- Increased code portability and ease-of-maintenance.
- Increased programmer productivity, both individually and across development teams.
- Reduced development and support costs.
- Extended code-life.
- Streamlined code reviews, which makes them more efficient and beneficial.
- Improved overall coding process efficiency by automatic generation of source code.

In order to **enforce the programming standards**, Verax Systems has a number of tools in place, such as:

- Standard formatting plugins and source code editors such as Eclipse or CDT.
- Static source code analysis plugins for Eclipse with Verax compliance rules for Java. For C and C++ code checking (linting) is built into the Verax build environment and makefiles. It can be performed by anyone at any time from the makefile.
- Code compliance checking integrated with the continuous integration system.

In addition, coding style compliance is checked at weekly code reviews. It is worth mentioning that Verax Systems relies on standards such as Sun's coding style for Java or Adobe coding style for Flex wherever possible.

6. Standardized development tools and virtualization

At Verax Systems for each project an approved set of tools is selected. In addition a reference VM*Ware virtual machine is created. Such an approach ensures that:

- Everyone can build or change anything on any workstation.
- Development environments are the same on all the workstations regardless of the operating system.
- Consistent builds are achieved across all software development workstations.

After employing the standard tools policy, a significant increase in engineering productivity has been observed due to a consistent, manageable and predictable environment.

The use of virtualization brings an additional benefit: developers can switch quickly among many projects (e.g. to examine an issue report) without having to make timely changes to their workstation.

7. Source code management, build process and continuous integration

Proper source code management, build process and continuous integration are a must for any modern software development. In this area, Verax Systems relies completely on industry-standard open source packages such as:

- **CVS** (<http://www.nongnu.org/cvs/>) for source code management. The standard CVS has been augmented with per-project security access control lists. CVS is coupled with the CVSSPAM notification system that reports all source changes by sending an e-mail to all interested parties. This mechanism allows on-line code reviews, which are more flexible and efficient than scheduled ones. The CVSSTAT package allows easy quantification of each engineer's work in terms of lines added or modified, number of commits and other metrics (these go later into the KPI dashboard – see section 9 for details).
- **Apache Maven** (<http://maven.apache.org/>) and **ANT** (<http://ant.apache.org/>) for Java builds and GMAKE for C/C++ ones. Maven also controls the release process, which at Verax Systems is similar to the one used by the Apache Software Foundation. For C and C++, the build environment is a sophisticated combination of GMAKE compatible makefiles and Perl scripts, that has been matured (first version originated around 1997) and proven in the field. The C++ build process supports:
 - Multiple target modes: debug/release, ANSI/Unicode and others.
 - Automated generation of source code dependencies.
 - Uniform, platform independent interfaces to popular tools including: flex, bison, strip and others.
 - Overnight builds of amended modules, source code documentation regeneration and CVS activity and source code statistics.
 - Integrated targets for source code documentation with Doxygen and JavaDoc documentation generation.
 - Source code statistics generation.
 - Integrated functions for test driver building, running and regression testing.
- **Cruise Control** (<http://cruisecontrol.sourceforge.net/>) for continuous integration for both C/C++ and Java. Cruise Control monitors the version control system for changes and automatically runs the build (i.e. Maven build) and the test process (i.e. JUnit testing). All detected bugs are immediately reported to project teams via e-mail.

Verax Systems uses POM (Project Object Models) for Java projects in order to manage releases of particular modules and check dependencies on modules used for the build (e.g. ensure that all developers build with the same version of log4j).

8. Test tools and environment

Testing at Verax Systems can be divided into two categories:

- **Module (unit) tests** (sometimes referred to as “programmer’s tests”) are executed on a per module basis by automated programs called test drivers. Module tests are closely integrated with the build environment and continuous integration.
- **System tests** (sometimes referred to as “user’s tests”) are executed against the entire system. System tests are usually more complex and difficult to automate. Often they require manual activities such as package installation and operator's interaction with the GUI. All system tests, which are not automated, have a test plan. The test plan contains a list of actions associated with proper pre- and post-conditions.

The Verax Systems module test drivers are CppUnit-based for the C/C++ languages and JUnit-based for Java. Test driver invocation is integrated with runtime analysis tools like Rational Purify Plus, Fortify, mpatrol, JProbe and others and with the regression mechanism, which allows that only tests affected by source code changes are run.

Automated system tests for web applications are executed using the Canoo (<http://webtest.canoo.com>) and Flex Monkey (<http://flexmonkey.org/>) frameworks.

Test drivers are designed in such way that module coverage is maximized (ideally 100% of code lines). Generated code (such as that produced by CORBA IDL compiler, bison and similar tools) is not included in the coverage requirement.

Verax Systems employs cross platform testing, which greatly improves code robustness – it is often a case that problems that are unseen on one platform appear on another one. The same holds true for testing tools and problems not reported by Purify can be detected by mpatrol. For Java, discrepancies arise mostly on various J2EE application server implementations (e.g. IBM, Sun or OpenSource such as JBoss).

Internally discovered bugs, enhancements and agile review results are tracked in Bugzilla (<http://www.bugzilla.org/>). Despite causing some overhead it has numerous advantages that outweigh its shortcomings. Its ability to generate statistics about modules causing most problems allows areas of weakness to be highlighted. Information about the change in the number of bugs between different versions enables tracking of the work progression. Employing Bugzilla also ensures that bugs and ideas for future improvements are captured.

9. Metrics, KPIs and continual improvement

Verax Systems uses two systems in order to calculate software development related metrics and KPIs: JCSC (<http://jcsc.sourceforge.net/>) and Verax KPI Dashboard (<http://www.veraxsystems.com/en/products/kpidashboard>). The first one is integrated with the continuous integration system (Cruise Control) and is responsible for calculating **source code related metrics** including:

- Test coverage statistics,
- Coding style compliance,
- CCN (cyclomatic complexity number),
- NCSS (non comment source statements),
- Unit test coverage,
- Source code duplicates and others.

The Verax KPI Dashboard controls metrics for the **entire software development process**, such as:

- Numbers of issues per project and trends (is the team improving?),
- Number of test cases per project.
- Developer productivity (lines of code produced) and others.

Some of these metrics are populated to the KPI Dashboard's data warehouse from external sources such as CVS Stat or Bugzilla.

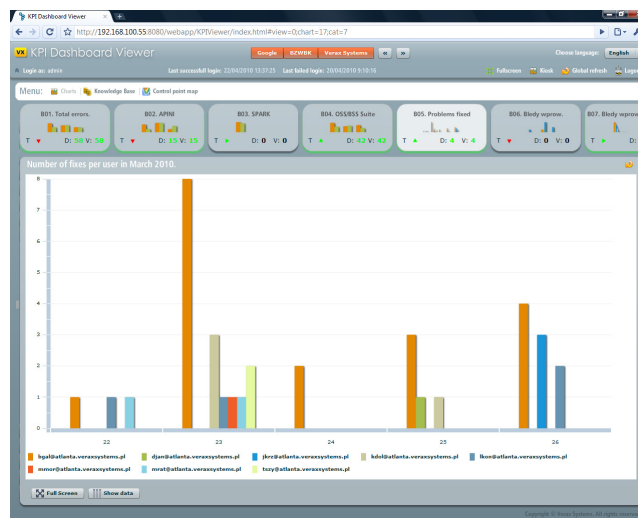


Figure 3: Software development KPIs.

The key benefits of having these metrics in place include:

- Evaluation of individual team members increasing overall code quality.
- Evaluation of project quality on a per-project basis.
- Detection of code that is error prone and/or costly to maintain in the future.
- Automatic detection of incompatibilities with coding and/or design standards.

For a client, this means improved value per dollar invested on software development.

10. Human factor

Software is created by humans. No matter how good the processes and procedures are, everyone can make a mistake. The challenge for each company is to train the staff in accordance to its needs and ensure that the knowledge stays within the company as employees rotate. Verax Systems has addressed these challenges by:

- Introducing a **knowledge base system** (integrated with corporate intranet) which contains solutions to frequently encountered issues, problems, HOWTOs, references to relevant sources, standards and other useful information.
- Intranet **project portals** (part of the Verax Project Portfolio Management System) accumulating most important project information (releases, build environment configuration, file shares, document libraries, etc.). Internal, intranet-based discussion groups are aimed at promoting the exchange of information between employees as well as ensuring that all information is archived for future use.
- Short but frequent **staff trainings**. Training material for all conducted training is available on the Intranet site for an easy reference.

A short (1-2 days) introductory training and assigning a mentor to each newcomer has given good results allowing new staff to adapt quickly and become productive in a short time.

11. Summary

The policies and practices outlined in the previous sections have helped Verax Systems to **deliver robust, high-quality, carrier-grade software** in many projects. The process that Verax Systems employs currently has evolved and matured over many years and has been extensively tested and verified on subsequent projects. Verax Systems is constantly improving its procedures as new platforms, tools and technologies are introduced. Our process-oriented approach enabled us to offer:

- Short project delivery cycles.
- Lower costs of carrier-grade system development.
- Superior software quality and technical excellence.

For more information please contact us or visit us on the Internet at <http://www.veraxsystems.com/en/services>.