Testing Safety-Critical Software

Testing safety-critical software differs from conventional testing in that the test design approach must consider the defined and implied safety of the software at a level as high as the functionality to be tested, and the test software has to be developed and validated using the same quality assurance processes as the software itself.

by Evangelos Nikolaropoulos

Test technology is crucial for successful product development. Inappropriate or late tests, underestimated testing effort, or wrong test technology choices have often led projects to crisis and frustration. This software crisis results from neglecting the imbalance between constructive software engineering and analytic quality assurance. In this article we explain the testing concepts, the testing techniques, and the test technology approach applied to the patient monitors of the HP OmniCare family.

Patient monitors are electronic medical devices for observing critically ill patients by monitoring their physiological parameters (ECG, heart rate, blood pressure, respiratory gases, oxygen saturation, and so on) in real time. A monitor can alert medical personnel when a physiological value exceeds preset limits and can report the patient's status on a variety of external devices such as recorders, printers, and computers, or simply send the data to a network. The monitor maintains a database of the physiological values to show the trends of the patient's status and enable a variety of calculations of drug dosage or ventilation and hemodynamic parameters.

Patient monitors are used in hospitals in operating rooms, emergency rooms, and intensive care units and can be configured for every patient category (adult, pediatric, or neonate). Very often the patient attached to a monitor is unconscious and is sustained by other medical devices such as ventilators, anesthesia machines, fluid and drug pumps, and so on. These life-sustaining devices are interfaced with the patient monitor but not controlled from it.

Safety and reliability requirements for medical devices are set very high by industry and regulatory authorities. There is a variety of international and national standards setting the rules for the development, marketing, and use of medical devices. The legal requirements for electronic medical devices are, as far as these concern safety, comparable to those for nuclear plants and aircraft.

In the past, the safety requirements covered mainly the hardware aspects of a device, such as electromagnetic compatibility, radio interference, electronic parts failure, and so on. The concern for software safety, accentuated by some widely known software failures leading to patient injury or death, is increasing in the industry and the regulatory bodies. This concern is addressed in many new standards or directives such as the Medical Device Directive of the European Union or the U.S. Food and Drug Administration. These legal requirements go beyond a simple validation of the product; they require the manufacturer to provide all evidence of good engineering practices during development and validation, as well the proof that all possible hazards from the use of the medical instrument were addressed, resolved, and validated during the development phases.

The development of the HP OmniCare family of patient monitors started in the mid-1980s. Concern for the testing of the complex safety-critical software to validate the patient monitors led to the definition of an appropriate testing process based on the ANSI/IEEE software engineering standards published in the same time frame. The testing process is an integral part of our quality system and is continuously improved.

The Testing Process

During the specifications phase of a product, extended discussions are held by the crossfunctional team (especially the R&D and software quality engineering teams) to assess the testing needs. These discussions lead to a first estimation of the test technology needed in all phases of the development (test technology is understood as the set of all test environments and test tools). In the case of HP patient monitors the discussion started as early as 1988 and continues with every new revision of the patient monitor family, refining and in some cases redefining the test technology. Thus, the test environment with all its components and the tools for the functional, integration, system, and localization testing evolved over a period of seven years. Fig. 1 illustrates the testing process and the use of the tools.

The test process starts with the test plan, a document describing the scope, approach, resources, and schedule of the intended test activities. The test plan states the needs for test technology (patient simulators, signal generators, test tools, etc.). This initiates subprocesses to develop or buy the necessary tools.

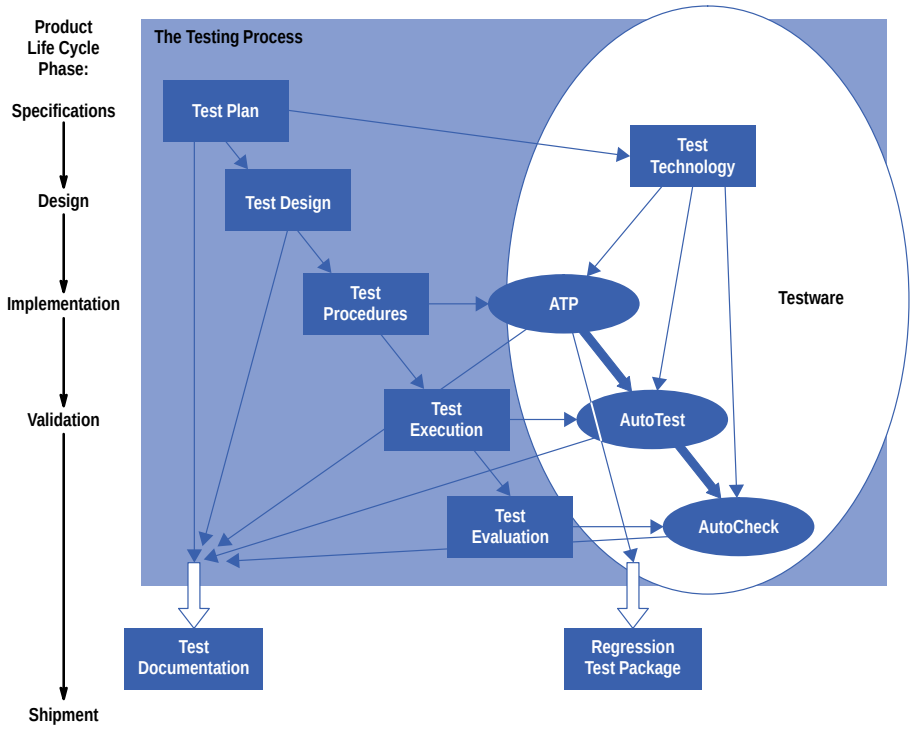

Fig. 1. *The software testing process for HP OmniCare patient monitors.*

Test design is the documentation specifying the details of the test approach and identifying the associated tests. We follow three major categories of test design for the generation of test cases (one can consider them as the main directions of the testing approach): white box, black box, and risk and hazard analysis.

The white box test design method is for design test, unit test, and integration tests. This test design is totally logic-driven and aims mainly at path and decision coverage. Input for the test cases comes from external and internal specifications (design documents). The test design for algorithm validation (proof of physiological measurement algorithms) follows the white box method, although sometimes this is very difficult, especially for purchased algorithms.

The black box test design method is for functional and system test. This test design is data-driven and aims at the discovery of functionality flaws by using exhaustive input testing. Input for the test cases comes from the external specifications (as perceived by the customer) and the intended use in a medical environment.

Risk and hazard analysis is actually a gray box method that tries to identify the possible hazards from the intended and unintended use of the product that may be potential sources of harm for the patient or the user, and to suggest safeguards to avoid such hazards. Consider, for instance, a noninvasive blood pressure measurement device that may overpump. Hazard analysis is applied to both hardware (electronic and mechanical) and software, which interoperate and influence each other. The analysis of events and conditions leading to a potential hazard (the method used is the fault tree, a cause-and-effect graph) goes through all possible states of the hardware and software. The risk level is estimated (the risk spectrum goes from catastrophic to negligible) by combining the probability of occurrence and the impact on health. For all states with a risk level higher than negligible, appropriate safeguards are designed. The safeguards can be hard or soft (or in most cases, a combination of both). The test cases derived from a hazard analysis aim to test the effectiveness of the safeguards or to prove that a hazardous event cannot occur.

Test cases consist of descriptions of actions and situations, input data, and the expected output from the object under test according to its specifications.

Test procedures are the detailed instructions for the setup, execution, and evaluation of results for one or more test cases. Inputs for their development are the test cases (which are always environment independent) and the test environment as defined and designed in the previous phases. One can compare the generation and testing of the test procedures to the implementation phase of code development.

Testing or test execution consists of operating a system or component under specified conditions and recording the results. The notion of testing is actually much broader and can start very early in the product development life cycle with specification inspections, design reviews, and so on. For this paper we limit the notion of testing to the testing of code.

Test evaluation is the reporting of the contents and results of testing and incidents that occurred during testing that need further investigation and debugging (defect tracking).

While test design and the derivation of test procedures are done only once (with some feedback and rework from the testing in early phases, which is also a test of the test), testing and test evaluation are repeatable steps usually carried out several times until the software reaches its release criteria.

Various steps of the testing process also produce test documentation, which describes all the plans for and results of testing. Test or validation results are very important for documenting the quality of medical products and are required by regulatory authorities all over the world before the product can be marketed.

The regression test package is a collection of test procedures that can be used for selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified product and legal requirements.

From the Test Plan to the Testware

Ten or fifteen years ago it was perhaps enough to give a medical instrument to some experts for clinical trials, algorithm validation, and test. The instrument had a simple display, information placement was not configurable, and the human interface was restricted to a few buttons and knobs. All the attention was on the technical realization of the medical solution (such as ECG monitoring), and software, mainly written by the electrical engineers who designed the hardware, was limited to a few hundred statements of low-level languages.

Today the medical instruments are highly sophisticated open-architecture systems, with many hundreds of thousands lines of code. They are equipped with complex interfaces to other instruments and the world (imagine monitoring a patient over the Internet—a nightmare and a challenge at the same time). They are networked and can be remotely operated. This complexity and connectivity requires totally new testing approaches, which in many cases, are not feasible without the appropriate tooling, that is, the *testware*.

Discussion of the test plan starts relatively early in the product life cycle and is an exit criterion for the specifications phase. One of the major tasks of the testing approach is the assessment of the testing technology needed. The term technology is used here in its narrow meaning of process plus hardware and software tools.

The testing technology is refined in the next phases (design and implementation) and grows and matures as the product under development takes shape. On the other hand, the testing tools must be in place before the product meets its implementation criteria. This means that they should be implemented and validated before the product (or subproduct) is submitted for validation. This requirement illustrates why the test technology discussion should start very early in the product life cycle, and why the testware has a "phase shift to the left" with respect to the product validation phase (see Fig. 2).

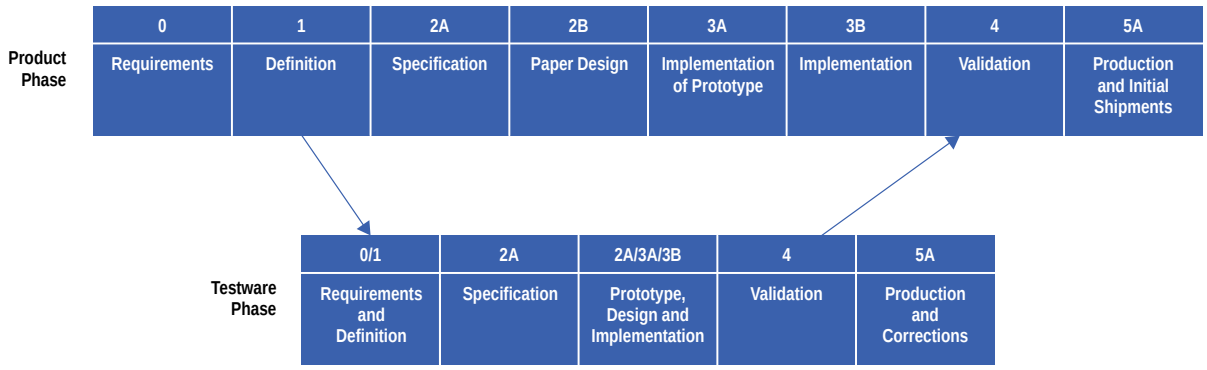

Fig. 2. *Relationship of product to testware development phases.*

Test Tool (Testware) Development

Testware development follows the same product life cycle as the product under development. The phases are:

- Requirements and Definition Phase. The test needs are explained according to the test plan and the high-level test design. Alternatives are discussed and one is selected by the software quality team.
- Specifications Phase. The tool is described in as much detail as possible to enable the testers to start work with their test cases and test procedures as if the tool were already available. These specifications are reviewed (or formally inspected) by the product development and test teams, approved, and put under revision control.
- Design and Implementation Phase. Emphasis is on the rapid development of engineering prototypes of the tools, which again are the object of review. These prototypes are used by the test team for low-level test design and first test trials.

- Validation Phase. The test tool is validated against its specifications. The most up-to-date revision of the patient monitor under development is used as a test bed for the tool's validation. Notice the inversion of roles: the product is used to test the test tool! Our experience shows that this is the most fruitful period for finding bugs in both the tool and the product. A regression package for future changes is created. First hardware construction is started if hardware is involved.
- Production Phase. The tool is officially released, hardware is produced (or bought), and the tool is used in the test environment. After some period of time, when the tool's maturity for the test purposes has been assessed, the tool is made public for use by other test departments, by marketing for demos, by support, and so on.

Fig. 2 demonstrates the main difficulty of testware development: the test tool specifications can be created after the product specifications, but from this point on, all of the testware development phases should be earlier than the product development phases if the product is to be validated in a timely manner.

Besides the shift of the development phases, there is also the testware dilemma: as the progress of the product's design and the test design leads to new perceptions of how the product can be tested, new opportunities or limitations appear that were previously unknown, and influence the scope of the testware. The resulting changes in the testware must then be made very quickly, more quickly than the changes in the product. Only the application of good hardware and software engineering processes (the tester is also a developer) can avoid having the wrong test tool for the product.

AutoTest

The test technology assessment for the patient monitors led us to the development of a number of tools that could not be found on the market. This make instead of buy decision was based mainly upon the nature of the patient monitors, which have many CPUs, proprietary operating systems and networks, proprietary human interfaces, true real-time behavior, a lot of firmware, and a low-level, close-to-the-machine programming style. Testing should not be allowed to influence the internal timing of the product, and invasive testing (having the tests and the objects under test on the same computer) had to be avoided.

The first tool developed was AutoTest,[1] which addressed the need for a tool able to (1) simulate the patient's situation by driving a number of programmable patient simulators, (2) simulate user interactions by driving a programmable keypusher, and (3) log the reaction of the instrument under test (alarms, physiological values, waves, recordings, etc.) by taking, on demand, snapshots of the information to send to the medical network in a structured manner.

AutoTest was further developed to accept more simulators of various parameters and external non-HP devices such as ventilators and special measurement devices attached to the HP patient monitor. AutoTest now can access all information traveling in the internal bus of the instrument (over a serial port with the medical computer interface) or additional information sent to external applications (see ***Article 14***).

AutoTest is now able to:

- Read a test procedure and interpret the instructions to special electronic devices or PCs simulating physiological signals
- Allow user input for up to 12 patient monitors simultaneously over different keypushers (12 is the maximum number of RS-232 interfaces in a PC)
- Allow user input with context-sensitive keypushing (first search for the existence and position of an item in a menu selection and then activate it)
- Maintain defined delays and time dependencies between various actions and simulate power failure conditions
- Read the reaction of the device under test (alarms, physiological values and waves with all their attributes, window contents, data packages sent to the network, overall status of the device, etc.)
- Drive from one PC simultaneously the tests of up to four patient monitors that interact with each other and exchange measurement modules physically (over a switch box)
- Execute batch files with any combination of test procedures
- Write to protocol files all actions (user), simulator commands for physiological signals (patient), and results (device under test) with the appropriate time stamps (with one-second resolution).

AutoCheck

The success of AutoTest and the huge amount of data produced as a result of testing quickly led to the demand for an automated evaluation tool. The first thoughts and desires were for an expert system that (1) would represent explicitly the specifications of the instrument under test and the rules of the test evaluation, and (2) would have an adaptive knowledge base. This solution was abandoned early for a more versatile procedural solution named AutoCheck (see ***Article 14***). By using existing compiler-building knowledge we built a tool that:

- Enables the definition of the expected results of a test case in a formal manner using a high-level language. These formalized expected results are part of the test procedure and document at the same time the pass-fail criteria.
- Reads the output of AutoTest containing the expected and actual results of a test.

- Compares the expected with the actual results.
- Classifies and reports the differences according to given criteria and conditions in error files similar to compiler error files.

AutoCheck has created totally new and remarkable possibilities for the evaluation of tests. Huge amounts of test data in protocol files (as much as 100 megabytes per day) can be evaluated in minutes where previously many engineering hours were spent. The danger of overlooking something in lengthy protocols full of numbers and messages is eliminated. AutoCheck provides a much more productive approach for regression and local language tests. For local language tests, it even enables automatic translation of the formalized pass-fail criteria during run time before comparison with the localized test results (see ***Article 15***).

ATP

The next step was the development of a sort of test generator that would:

- Be able to write complex test procedures by keeping the test design at the highest possible level of abstraction
- Enable greater test coverage by being able to alter the entry conditions for each test case
- Eliminate the debugging effort for new test procedures by using a library of validated test primitives and functions
- Take account of the particularities and configurations of the monitors under test by automatic selection of the test primitives for each configuration
- Produce (at the same time as the test setup and the entry conditions) the necessary instructions for automated evaluation by AutoCheck.

The resulting tool is called ATP (Automated Test Processor, see ***Article 13***). Like AutoCheck, ATP was developed by using compiler-building technology.

Results

Good test design can produce good and reliable manual tests. The industry has had very good experience with sound manual tests in the hands of experienced testers. However, there is no chance for manual testing in certain areas of functionality such as interprocess communication, network communication, CPU load, system load, and so on, which can only be tested with the help of tools. Our process now leaves the most tedious, repetitive, and resource-intensive parts of the testing process for the automated testware:

- ATP for the generation of test procedures in a variety of configurations and settings based on a high-level test design
- AutoTest for test execution, 24 hours a day, 7 days a week with unlimited combinations of tests and repetitions
- AutoCheck for the automated evaluation of huge amounts of test protocol data.

One of the most interesting facets is the ability of these tools to self-document their output with comments, time stamps, and so on. Their output can be used without any filtering to document the test generation with pass-fail criteria, test execution with all execution details (test log), and test evaluation with a classification of the discrepancies (warnings, errors, range violations, validity errors, etc.).

Automated testware provides us with reliable, efficient, and economical testing of different configurations or different localized versions of a product using the *same* test environment and the *same* test procedures. By following the two directions of (1) automated testware for functional, system, and regression tests (for better test coverage), and (2) inspection of all design, test design, and critical code (as identified by the hazard analysis), we have achieved some remarkable results, as shown in Fig. 3.

Through the years the patient monitor software has become more and more complex as new measurements and interfaces were added to meet increased customer needs for better and more efficient healthcare. Although the software size has grown by a factor of three in six years (and with it the testing needs), the testing effort, expressed as the number of test cycles times the test cycle duration, has dropped dramatically. The number of test cycles has dropped or remained stable from release to release.

The predictability of the release date, or the length of the validation phase, has improved significantly. There has been no slippage of the shipment release date with the last four releases.

The ratio of automated to manual tests is constantly improving. A single exception confirms the rule: for one revision, lack of automated testware for the new functionality—a module to transfer a patient database from one monitor to another—forced us to do all tests for this function manually.

The test coverage and the coverage of the regression testing has improved over the years even though the percentage of regression testing in the total testing effort has constantly increased.

See Subarticle 12a: ***Processor and System Verification***.

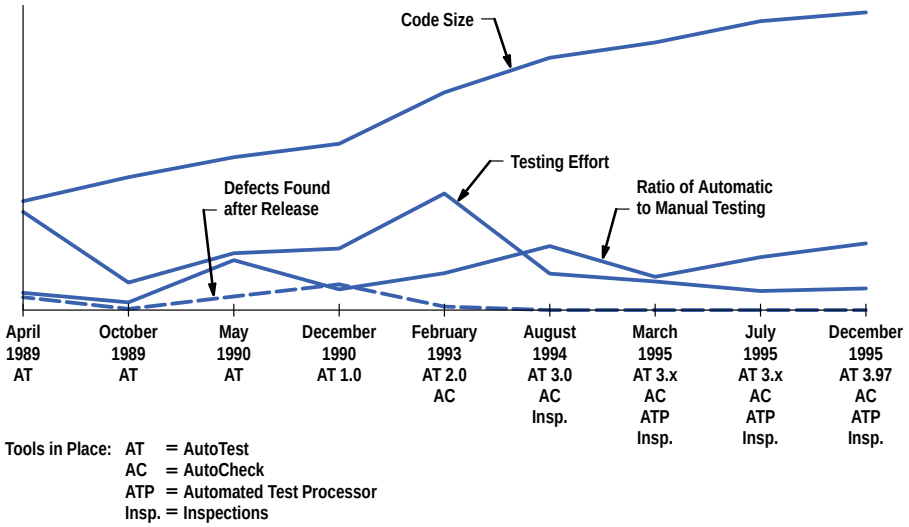

Fig. 3. *Trends of testing metrics for patient monitors.*

Conclusion

Software quality does not start and surely does not end with testing. Because testing, as the term is used in this article, is applied to the final products of a development phase, defect discovery through testing always happens too late in each phase of product development. All the experience gained shows that defect prevention activities (by applying the appropriate constructive software engineering methods during product development in all phases) is more productive than any analytic quality assurance at the end of the development process.

Nevertheless, testing is the ultimate sentinel of a quality assurance system before a product reaches the next phase in its life cycle. Nothing can replace good, effective testing in the validation phase before the product leaves R&D to go to manufacturing (and to our customers). Even if this is the only and unique test cycle in this phase (if the defect prevention activities produced an error-free product, which is still a vision), it has to be prepared very carefully and be well documented. This is especially true for safety-critical software, for which, in addition to functionality, the effectiveness of all safeguards under all possible failure conditions has to be tested.

In this effort, automated testware is crucial to ensuring reliability (the testware is correct, validated, and does not produce false negative results), efficiency (no test reruns because of testware problems), and economy (optimization of resources, especially human resources).

Reference

1. D. Göring, “An Automated Test Environment for a Medical Patient Monitoring System,” *Hewlett-Packard Journal*, Vol. 42, no. 4, October 1991, pp. 49-52.

▶ Go to Subarticle 12a
▶ Go to Next Article
▶ Go to Journal Home Page