Software Engineering and Management

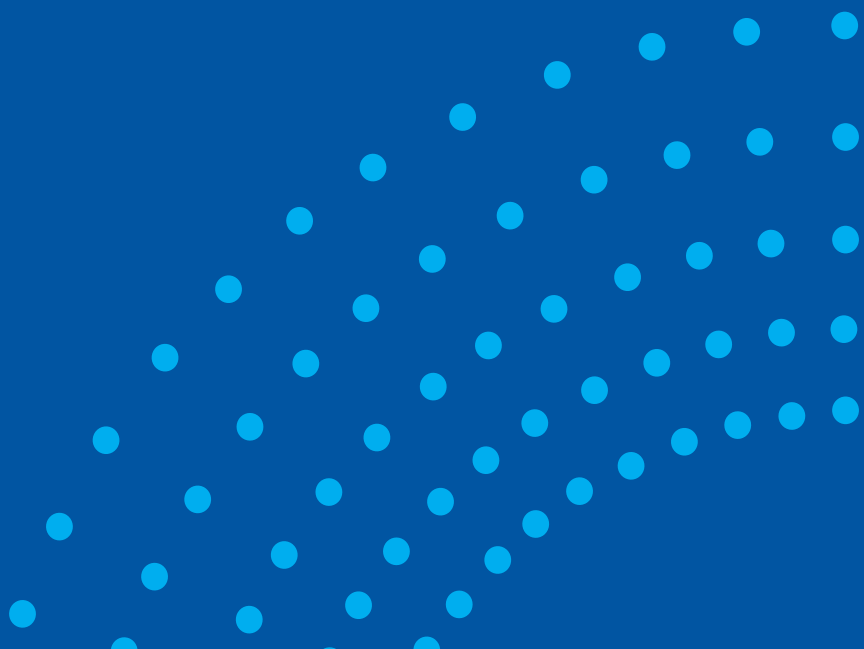
Testing a radiotherapy support system with QuickCheck

Aiko Fallas Yamashita
Andreas Bergqvist

Göteborg, Sweden 2007

IT University of Göteborg
CHALMERS | GÖTEBORG UNIVERSITY

Master Thesis Report



REPORT NO. 2007/62

Testing a radiotherapy support system with QuickCheck

Master Thesis Report

AIKO FALLAS YAMASHITA

ANDREAS BERGQVIST

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2007

Testing a radiotherapy support system with QuickCheck

Master Thesis Report

AIKO FALLAS YAMASHITA

ANDREAS BERGQVIST

Report no 2007:62

ISSN: 1651-4769

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

P O Box 8718

SE – 402 75 Göteborg

Sweden

Telephone + 46 (0)31-772 4895

Göteborg, Sweden 2007

The current document presents the results from the Masters Thesis work carried out during the Spring Term 2007 in the Software Engineering and Management Program.

Title: Testing a radiotherapy support system with QuickCheck

Aiko Fallas Yamashita

Andreas Bergqvist

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

Supervisor: Thomas Arts

SUMMARY

In this report we present a case study on using light-weight formal methods for testing an implementation of a medical device. The device is a real-time organ position tracking system used in radiotherapy. The system properties to be tested were modeled and verified through automated test cases generated by QuickCheck. QuickCheck demonstrated to be beneficial tool for reducing the complexity inherent to testing medical devices by detecting faults at system level, supporting a better type of regression testing and supporting the detection of abnormal cases that could be analyzed and fixed afterwards in the system.

Keywords: Light-weight formal methods, medical devices, software verification, software testing.

Table of Contents

1. Introduction

Failures in medical devices can have dramatic consequences by putting peoples' health or even life at risk. With the increase of software use in these devices, there are high demands on software verification and analysis in the medical domain. In many cases formal methods are used to model medical devices [7], medical protocols [3], or even complete systems [13]. No matter how well one has formally proved correctness of a model of a medical device, its implementation still needs to be tested. Testing medical devices is normally done very thoroughly [11] as demanded by organizations like the FDA and its European counterpart, MDD.

In this article we present a case study on using light-weight formal methods for testing an implementation of a medical device. The device is a real-time organ position tracking system used in radiotherapy, i.e., a tumor is positioned in real-time in order to be able to accurately deliver a dose of radiation. The device is constructed by Micropos Medical, a Swedish start-up company.

The positioning system has a clear mathematical model, viz., a three dimensional space with an additional time axis. Tumor and radiation centers are two of a hand full of important points in this space. The distance between the estimated position and actual position of the tumor must lie in a range of 2 millimeters. The system was tested by using QuickCheck [6] provided by the company Quviq. QuickCheck is a testing tool that randomly generates test cases from a given model. By specifying a model of the coordinate system we were able to effectively use QuickCheck to generate thousands of tests. By doing so, we have identified some failures caused by incongruence in the hardware drivers specification, misinterpretations of the pseudo-code (i.e. declaration of global variables and static values interpreted as local and dynamic variables), inadequate handling of floating point operations, wrong type conversion, which have been successfully fixed. It was possible to identify from a set of prototypes of the underlying mathematical model, the ones which provided unacceptable accuracy and it was possible to discern which ones provided the best accuracy.

The rest of the report is organized as follows: Sect. 2 introduces the context on medical device verification, bringing up the need of integrating test tools and formal methods in the medical industry, presenting also an overview of the 4DRT system and how QuickCheck was used in order to test it. Sect. 3 provides a brief description of the research methodology used in this study. Sect. 4 provides a more in-depth description of the case study, indicating the testing setup, the properties tested in the system and the implementation process. Sect. 5 describes the results and analysis derived from this study, finalizing with Sect. 6, which presents the conclusions for this study.

2. Verifying medical devices

The process of delivering medical devices encompasses mainly two approaches: process centered verification and artifact centered verification [12]. Process centered verification is often described by standards that suggest a series of practices for the medical practitioners to base the development of the safety-critical software products on [21], [5]. However, there is a visible need for more artifact centered approaches as medical devices turn more and more sophisticated, complex and wide-ranging; and general guidelines are proving to be insufficient for delivering a safe product [12], [10]. Moreover, regulations from medical authorities as the FDA or MDD are starting to move from process based certification to prove based certification [16].

The application of formal methods in medical devices supports this line of action and could add significant confidence in the system by revealing errors in both the system's model and its implementation [4]. There are many success stories of using formal methods in the medical domain, which ranges from medical protocols [3], [15] to medical equipment controllers [9], [13] and medical devices [**7**]. Studies point out the need of well established processes that include formal methods in order to ensure safety systems [19].

Despite the numerous advantages of formal methods, there is still a need for testing the actual implementation given the differences that could exist between the model and the implementation. Furthermore, a testing process is a "must" in current certification processes regulated by medical authorities [11], [8], [14]. The development of testing frameworks/tools and techniques which support adequately formal methods in the pursuit of safe systems is still an open problem. One of the biggest challenges pointed out by [16] will be the incorporation of modeling techniques and practical formal methods to the design of software-based medical systems in the future.

As the need of practical cases to support the usage of formal methods in medical systems increases, it is possible to learn from other domain's experiences in the use of technologies and approaches in practical situations. We looked at a novel successful approach in the telecommunications domain with software verification through a combination of formal methods and testing [2].

2.1. Position tracking device

4DRT (Four Dimension Radio Therapy) is a real-time organ position tracking system optimized for the human body. It can give the position of the organ in four dimensions, a three dimensional space and time. This enables monitoring of the position of tumors in cancer patients and thereby helps to improve the accuracy of the radiation during radiotherapy treatments.

The system is based on radio frequency transmission. The measurement of the position is done through an implantable device in the organ (or nearby), which acts as a transmitter. The transmitter emits a radio frequency which is captured by multiple receiving antennas or receivers, typically arranged in a plate on the treatment table under the patient (See Fig. 1). The signal captured by the receivers is sent to the core system where it is used to calculate the position of the organ.

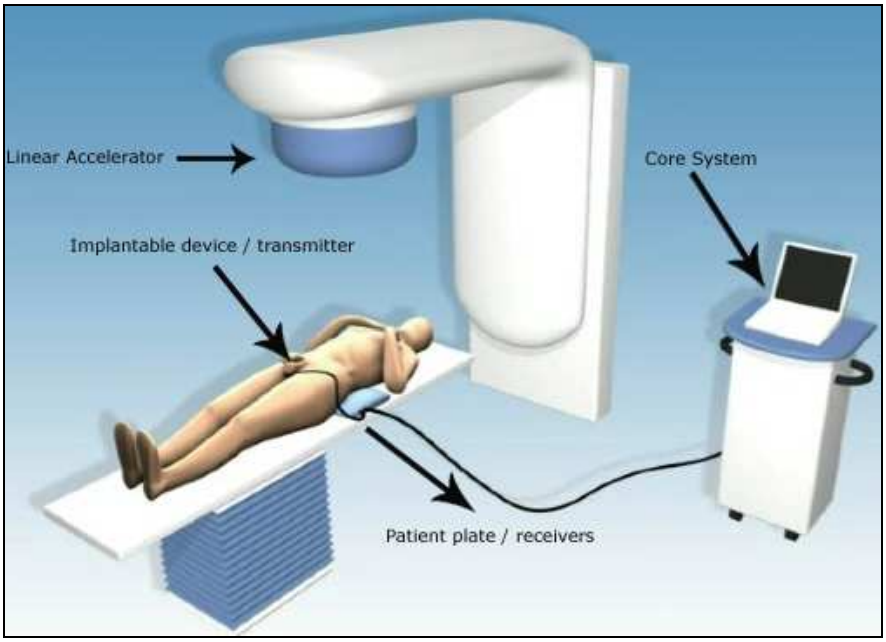


Fig. 1. View of 4DRT system in a treatment environment. Elements such as the Linear Accelerator, the implantable device, the patient plate and the 4DTR system are depicted.

Both transmitter and receivers are connected to the hardware controller, which is here treated as a black box for both hardware control and signal processing. The hardware controller contains an analog-digital converter that will continuously send a predetermined set of decimal values (representing the measured signal) to the core system through a USB port. The core system is the software component in the 4DRT system that performs the position computation by mapping the decimal values received from the hardware controller to a specific coordinate position in the real world. This coordinate position is given in the 4DRT coordinate system which have a predetermined range for each axis (X, Y, Z) and the angles (Vy, Vz), i.e., rotation over y and z axes. The core system also handles the interaction with external systems and the user interface.

A mathematical model is used to calculate (based on the received radio signals) an estimated position. This model should guarantee that an estimated position corresponds close enough to the real position. This position calculation is the main feature to be tested and the main topic of this paper. We want to make sure that the positions calculated by the system are within a radial distance of 2mm from the actual position (this to ensure that the tumor receives radiation and not the healthy tissue around it). This needs to be done extensively with thousands of different positions, since it is far from enough to measure only once and let that represent the accuracy of the system. Of course, it is rather

easy to randomly generate positions that we want to test. We used the tool QuickCheck to do this and in addition it executed the test and observed whether the measured value was indeed within the demanded range. The advantage of QuickCheck over other tools here is that the input is very close to the mathematical specification that one would expect (hence easy to inspect that the right aspect of the system has been tested). Another advantage is the automatic simplification of failing test cases that QuickCheck provides, which will be explained in more detail later.

Note that we propose a kind of system test from the beginning. Given the simplicity of the property that we want to test and the dependency of the whole system to correctly pass a large number of tests, we estimate that we can catch all failures that otherwise would be caught by unit testing. Thus, this seems a case where starting with system testing and leaving out unit testing is cheaper than designing dedicated tests for each unit. The lab setting used during testing consisted of Transmitter, Receiver, Hardware Controller, Core System and Auto Setup (a mechanical device able to move the transmitter to specified coordinates X, Y, Z, Vy and Vz).

QuickCheck generated coordinates which were in the range supported by the mathematical model. The coordinates were then used to control the Auto Setup into moving the transmitter to the corresponding position. The position estimator calculated the position of the transmitter and "sent" the X, Y, Z, Vy, Vz coordinates back to QuickCheck. QuickCheck then determined the radial distance (in Euclidean space) between the generated position and the measured position by using Formula (1). A test fails if this distance is more than 2mm.

$$\sqrt{\left((X_p - X_c)^2 + (Y_p - Y_c)^2 + (Z_p - Z_c)^2\right)} \quad \textbf{(1)}$$

2.2. QuickCheck

QuickCheck is a tool that combines random test generation, with a flexible language for specifying generators and the use of properties to adjudge success [6]. In our case study we used the commercial QuickCheck tool provided by Quviq that offers automatic simplification of the test cases when a failure has been detected, i.e., the property was refuted. This simplification is handy when removing the inevitable "noise" that random testing brings.

For example, a property to test whether received coordinates are within a margin would look as follows in the QuickCheck specification language (which is actually the functional programming language Erlang [1] :

```
prop_within_margin(Margin) ->
  ?FORALL(Coordinate, antenna_coordinate(),
          begin
            move_antenna_to(Coordinate),
            Position = read_position(),
            radial_distance(Position, Coordinate) =< Margin
          end).

radial_distance({XP,YP,ZP},{XC,YC,ZC}) ->
  math:sqrt(
      math:pow(XP-XC,2)+math:pow(YP-YC,2)+math:pow(ZP-ZC,2)).
```

In this example `antenna_coordinate()` is a function that generates a random triple of x, y and z coordinates and a fourth value, which is the angle of the antenna relative to the specific surface. The result of generation is bound to the variable `Coordinate`. QuickCheck generates by default hundred tests for each such property, thus hundred tests are generated which each their own randomly generated coordinates. Each test case consists of three steps. First the antenna is moved to a certain position (see Sec. 4.1 for details on our test set-up). The function called `move_antenna_to(Coordinate)` returns a value when the antenna has reached the desired point. After that the most recently read position is fetched from the system. This position is then compared to the actual coordinates.

Whenever a test fails, i.e., any of the actions fails or the result of the last inequality is *false*, then QuickCheck will automatically search for a smaller counter example.

3. Research methodology

For this study, Action Research (AR) [18] was used as the research methodology. Action Research is research through a practical collaboration between researchers and industry practitioners, in order to solve a problem and striving to improve their strategies, practices, and accumulate knowledge of the environments in which they practice.

The immediate problem for Micropos is to create a fully functional medical product ready for release. The existing system is a prototype and the process necessary to certify and launch a commercial version of the system is supported by testing and formal methods. The company learns about the importance of adequate testing to ensure high quality, novel tools and methods and state-of-the-art in research, whereas at the same time, the researcher learns about the problem area, the state-of-the-art in industry and the difficulties in applying techniques in a practical situation.

Our research interest of trying QuickCheck within the medical domain can be feasibly combined within the context and needs of Micropos. At the same time this enables a better understanding of the role of formal models and *testware* within the medical software development. Furthermore, working in a real-world company provides valuable insights on how a certain technique/tool can be used in a given domain and the adequacy of such a technique to the domain. This is difficult to achieve within a solely academic environment.

Avison points out the need of system analysts to understand the organizations in terms of people's conflicting objectives, perceptions and attitudes and adds that failure to include human factors may explain some of the dissatisfaction with conventional information systems development methodologies; they do not address real organizations [23]. By using Action research we will try to bring up to discussion the relevance and impact of our study in the industry.

3.1. Roles of the researcher within the project

The main roles carried out as researcher during the project were:

Planner/Leader: In terms of the development and testing of the system, the planning and execution of the activities is the primarily role.

Designer, Developer and Tester: The researcher must work as an analyst, developer and tester the in the project

Observer/Reporter: This role is necessary in order to collect and document the outcomes and knowledge acquired from the practice, as stated in the Action Research framework.

In most Action Research situations, a researcher's role consists mainly in facilitating dialogue and promoting reflective analysis among the participants, providing them with periodic reports, and writing a final report when the researcher's involvement has ended [24].

3.2. Research Process

The Action Research model proposed by Kemis was used in order to perform the study (see Figure 1).

The study was based on a previous study (Research course) where software architecture for the system was developed. This can be visualized as the first cycle of the research where the results where assessed and a new action plan was proposed in coordination with the industry and research interests of the participants. The main outcomes from the first cycle were the understanding of the socio-technical environment, a clear picture of the main challenges and needs of Micropos and the material for continuing with the second cycle.

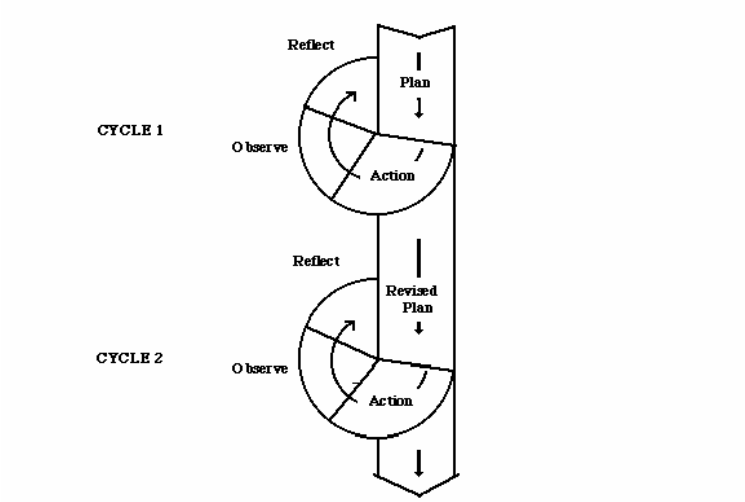


Fig. 2. Action Research Model proposed by Kemmis [25]

4. Case study

This case study strives to perform testing of 4DRT. But how can QuickCheck support this process? From a Risk Based Analysis point of view, we determined that verifying the accurate position calculation is the key importance for assuring a safety treatment delivery. A QuickCheck property was formulated in order to enable a formal model that can be corroborated through execution.

In terms of testing coverage, it is clear that due to the nature of the software we are testing, the process of requesting only one single position calculation will cover the critical path of the modules. Thus, if we move the transmitter to {0,0,0,0,0} and then we request the position, we would have full coverage without revealing any failure in the system. Therefore we need many data points to test. QuickCheck could provide an enough varied set of cases that can reveal errors in the software.

From a Black Box testing view, in this specific system, the testing has a relatively simple functional testing specification. The problem is in the amount of variation of the parameters, making the testing space very big (if we consider that the testing space is in mm and we have five independent parameters in the space, given reasonable finite floating point accuracy, the number of total testable points will ascend to billions). QuickCheck provides random testing which constitutes a feasible option for reasonably covering the testing space.

4.1 Testing setup

When starting the project, a prototype of the core system had already been produced in LabView and it had been tested by a semi-automatic method, i.e., manually specifying a list of points the transmitter should move to and then comparing this with the values the core system produced. It was required to implement the core system in C# on the .NET framework platform, based on the LabView version; and afterwards test the implementation.

Given the kind of the code we were going to develop, the requirements to be tested (Table 1), the simple underlying formula for correctness (Formula 1 in Sect 2.1), the ease with which this formula can be expressed in QuickCheck, and the kind of errors we can expect (typical for floating point handling), we decided to use system testing as the only way of testing. Nevertheless, some unit testing was unavoidable in the form of “experimenting” while migrating LabView to .NET.

We used a hardware setup in which the antenna can be moved to a specific point in a three dimensional space and in which it can be moved in an angle around the Y and Z axes. When acknowledged that the antenna had reached a certain position, the system test was performed and the estimated position was measured. We made sure that this was all within the timing requirements of the “fourth dimension”, i.e., the fact that the system should be able to track the antenna in real-time. The set-up is depicted and summarized in Fig. 3.

Table 1: Description of the functional requirement (and its correspondent non-functional requirement) tested in the system

Functional requirement	The software component should calculate the 5D positioning of the transmitter (X, Y, Z, Vy, and Vz, where Vy is rotation around Y axis and Vz is rotation around Z axis)
Non-functional requirement	The system should achieve 3D difference or radial accuracy of 2 ±1 mm

Each test was randomly generated by QuickCheck from a QuickCheck property similar to the one presented in Sect. 2.2. Typically, integer values specifying millimeters were used to move the antenna to a given position (the hardware moving the antenna supported ±1mm precision). For example, QuickCheck could generate from the property a test in which the antenna is steered to position: X=58, Y=127, Z=94, Vy=0, Vz=0, after which the estimated position: X=58.15106462, Y=126.9147189, Z=94.82734652, Vy=-2.582979671, Vz=-3.070729491 is registered. The distance to the real value is computed: 0.84533759 and since it is less than 2mm, the test passes successfully[1].

QuickCheck uses a uniform distribution in its random generation of coordinates. For the purpose of testing the software, we are satisfied by that. The non-functional requirement in Table 1 does indicate, however, to use a normally distributed set of sample points and to generate a normally distributed sample from them. Since patient data is unavailable at this point, we decided to be stricter than that and use a uniform distribution, requiring an accuracy of 2mm, without leaving space for points in one standard deviation. We analyzed the few failing tests (i.e., those with a distance larger than 2mm) to see how much they were off.

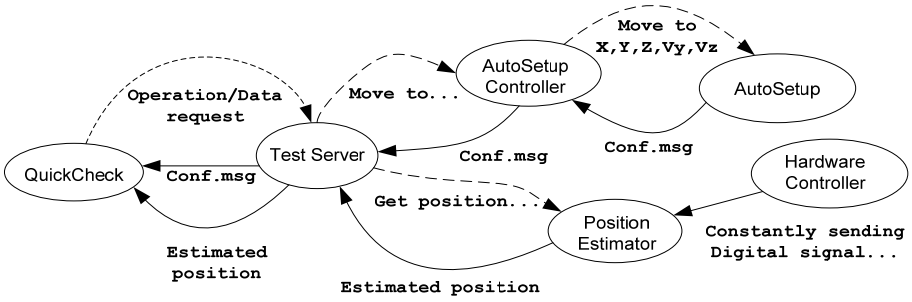


Fig. 3. Activity flow diagram describing the interaction between the QuickCheck module and the C# software components

QuickCheck communicated via TCP/IP with a sort of request broker that we implemented in C#. This broker (called Test Server, as depicted in Fig. 3) receives requests from QuickCheck and calls components in the Core System which are implemented in C#.

[1] It is important to point out that the Vy and Vz are only considered for performing the position calculation, but not for computing the radial distance. Hence the radial distance shown in the example only contemplates X, Y and Z.

4.2 Symmetry Property

As the testing process evolved, we found the need for testing one additional property, not presented in the requirements, but an important assumption about the software that was orally communicated to us.

The system works under the assumption that the mathematical model is symmetric, which means that one coordinate gives a similar accuracy on its correspondent extrapolated value (i.e. {36,41,73} gives similar results in accuracy as {134,139,171}). An example of code is presented in property `prop_symmetric()`.

```
prop_symmetric(Margin)->
  ?FORALL(Coordinate,antenna_coordinate (),
          begin
              Extrapolated = extrapolate(Coordinate),
              move_antenna_to(Coordinate),
              Pos1 = read_position(),
              move_antenna_to(Extrapolated),
              Pos2 = read_position(),
              Distance1 = radial_distance(Coordinate, Pos1),
              Distance2 = radial_distance(Extrapolated, Pos2),
              abs(Distance1 – Distance2) =< 1
          end).

extrapolate(Coordinate)->
    X = upper_x - abs(lists:nth(1,Coordinate)-lower_x),
    Y = upper_y - abs(lists:nth(2,Coordinate)- lower_y),
    Z = upper_z - abs(lists:nth(3,Coordinate)- lower_z),
    [X, Y, Z].
```

In the above code, we verify that the accuracy distance between a given coordinate value and its correspondent extrapolated coordinate is less than 1mm. We used 1mm as the delimitation value for practical reasons, which was experimentally determined by a test simplification where we could find the biggest distance between accuracies of extrapolated values in the system.

4.3 Implementation

In order to perform the migration of the software from LabView to .NET/C#, a bottom-up development process was followed by developing the smaller pieces of the code (i.e. the modules that handle the hardware drivers) and start building upon those. Pseudo-code documentation describing the algorithms used, as well as internal documentation and the source code of the prototype in LabView were used as supportive material for performing the migration.

The core system contains approximately 3000 lines of C# code divided amongst 8 components. It has 5 concurrent processes which perform tasks such as monitoring the hardware and retrieving the values of the measured radio signals.
The different components handle the communication with hardware drivers; perform mathematical computation (i.e. matrix, vector and polynomial operations), system logging, file access, network communication, etc. The Position Estimator component contains all the high level algorithms and calls the other components in order to perform the position calculation.

5. Results and Analysis

In the project, we spent officially 4 days in testing, without counting additional time for fixing bugs, writing changes in the properties, and other tasks. (We have to mention that during the project, use of the equipment was time restricted). In this section, results from the testing process, system specific results, perceived benefits and possible improvements are presented and discussed.

5.1 Test results

Most of the failures were detected in the first test cases QuickCheck produced from the main property described in Sect. 2.2. In all cases, it was possible to trace the failures back to the code so adequate corrections could be performed. Typical problems that involved floating point operations, type conversion, and use of wrong types in the drivers' interfaces were found. For instance, we found out that the hardware driver of the Auto Setup did not accept decimal points as parameters in one of the interfaces. This problem was identified when using QuickCheck for sending the coordinates to the Auto Setup and it was observed that the latter did not move the transmitter as expected. We could trace this problem back to a division operation performed prior to sending the coordinates to the hardware driver. This division produced decimal values occasionally instead of just integers. So the Auto Setup only moved when the resulting division was a whole number.

Another problem was the use of incorrect casting operations (i.e. truncating decimals instead of rounding) which was detected while observing a set of failed test cases which showed a very similar radial distance. We found out that conversion in LabView is implicitly managed, in contrast of C#, which requires a specific conversion method.

Also, errors due to misinterpretations of the pseudo-code (i.e. declaration of global variables and static values interpreted as local and dynamic variables) could be detected by observing failed test cases that showed a very big radial distance. A similar error was found in the same test cases, where we used an incorrect constant value for one of the algorithms (due to the fact that we were using an outdated version of LabView code for a specific module).

The already mentioned issues are typical when performing migration from two different platforms (in this case from LabView to C#), where some assumptions (i.e. typing, management of decimal values, etc) in the old platform are not longer valid in the new platform. They are also related to a typical situation in the medical domain when specifications regarding interfaces for hardware drivers as well as software COTS (components off the shelf) are not so clear [20]. QuickCheck facilitates code refinement and simplifies the task of detecting those errors (mostly within a couple of property executions).

Note that we found all these errors by specifying just one property and generating random test cases from them. Therewith, the work in creating test cases is dramatically simplified in contrast to more traditional testing approaches.
It was also possible to determine which mathematical models support a given radial accuracy. Whenever a new model is introduced, it is possible to test it with QuickCheck

and its adequacy would be visible almost immediately. For instance, one day we had introduced a model which was stated to have better accuracy than the previous one; we ran QuickCheck on it and found a coordinate with an unacceptable accuracy. It turned out that even in the prototype implementation with this newer model, the estimated point had unacceptable distance to the position. Hence, the model was further improved before introduced again.

Issues in the communication protocol between QuickCheck and C# were also detected with QuickCheck. For instance, a problem due to the overwriting of instructions into the hardware driver was detected while running the property (this overwriting issue resulted in a series of incomplete test executions). We found that the driver we were using for the hardware demands a lapse of 40ms in order to process one instruction and read the next one. Although the communication with QuickCheck and C# is not part of the system, this last example gives us an insight of how QuickCheck could support integration testing as well, where the communication between software components needs to be verified.

5.2. Perceived benefits

From this case study we could identify several key benefits of using QuickCheck in the testing of medical devices. These are described in the following subsections.

5.2.1. Better than Regression testing

We perceived that it was possible to introduce changes in other parts of the system (i.e. hardware, since this product is evolving constantly; making devices smaller, faster, etc) and afterwards use QuickCheck to perform high level testing. This enabled us to detect any incongruence or errors that might result as a consequence from those changes.

The same situation can be applied to code enhancements that we performed in order to improve performance (some algorithms in LabView could be implemented in C# in a more efficient way), and we could always run QuickCheck and make sure that these enhancements in the code give the same results as the original algorithms written in LabView. Furthermore, the coverage of the side-effects of changes introduced in the software (or hardware) is greater with QuickCheck since it generates new random test cases each time. In that sense, QuickCheck constitutes a good asset for a system that is constantly evolving, (a scenario very typical in medical device development [17]) by providing better support than regression tests which concentrate on the same tests over and over.

5.2.2. Improving the system quality

An example of how QuickCheck helped in improving the quality of the software was when we tried different mathematical models to see which ones gave better results (as explained previously in Sect. 5.1). Furthermore, having a formal specification of the system that can actually be run and corroborated constitutes a significant advantage for certification processes (as pointed out by [16] and mentioned in Sect. 2).

5.2.3. Cost effectiveness

Faults related to testing the mathematical model (accuracy checking) were detected after 12.85 test cases on average, and abnormal cases were detected after 78 tests on average. It is very unlikely that one would manually write test cases with the same averages, but it shows that several cases would have to be written for a good test suite, whereas here we only write one property once.

Each time a test case is run, the transmitter must be positioned before performing the measurement, and this is a rather expensive task if is not supported by an automated tool. In our case, it took in around 5-7 seconds per each test case, depending of the position the Auto Setup was moved to. QuickCheck requires relatively less amount of effort, and supports repetitiveness and generation of new values every time, so it was very useful for the type of testing performed.

5.3 Interesting findings regarding the system

While performing the testing of the system, some interesting facts regarding the system were found. Due to the explorative nature of this study, it is possible to encounter a diverse set of unplanned findings. Some of them are presented in the following subsections.

5.3.1. Coverage-range of mathematical model

The mathematical model establishes a limit (coverage-range) for providing acceptable measurements. Through QuickCheck we could confirm that these borders are non-inclusive (which means that if at least one of the axes of the coordinate system touches the limit border, it won't provide the accuracy required). In most of the cases we found that once that the transmitter reaches at least one of the limits of the coverage-range in any of the axes, the radial distance will always exceed 2mm. The generator in QuickCheck was modified to be non-inclusive, resulting in successful executions of 100 test cases each with an accuracy of 2mm.

5.3.2. Symmetry of the mathematical model

The system works under the assumption that the mathematical model is symmetric, which means that in one coordinate, the system gives distance similar to its corresponding extrapolated coordinate. This was confirmed by using QuickCheck with a new property, as described previously in Sect. 4.2.

5.3.3. Detection of abnormal cases

Sometimes you want to run the property for a longer period and use atypical test parameters in order to find abnormal cases. By extending the margin tolerance (increasing the radial accuracy limit), we could detect abnormal cases related to the transmitter angles (angles very close to the negative or positive borders gave significantly big radial distances). For instance, we ran the property with radial accuracy of 6mm. An apparently normal position (in the sense of that it was within the coverage-range) resulted in a radial distance of almost 6mm. After some more tests we found out that the mathematical model we were using in testing was sensitive to strongly angled positions (in terms of Vy and Vz), this finding lead to adjustments in the mathematical model in

order to improve its robustness against angling. We must mention that the parameters used for performing this type of testing exceeds the limits of what could be called a normal scenario (i.e. test parameters derived from real patient data) and abnormal cases rarely occur when actually using the system. Even with these considerations, the use of atypical parameters is still useful for improving the overall quality of the software, and this is a good example of such a case.

5.4. General remarks

Additionally to the information provided by QuickCheck, the logs in the Test Server side were useful to analyze all the test cases generated by QuickCheck (failed tests and successful tests) and observe the test cases simplification behavior as well as the radial distance from each of the measurements.

Within the given coverage range, the system provides even better accuracy than that specified by the non-functional requirement. One large sample of generated tests had a mean of 1.528505mm for the radial distance, with a standard deviation of ±0.477921, where 87% of test cases passed and 13% failed; from the failed test cases, 2% had between 2.4mm and 3.4mm of radial distance and 11% between 2mm and 2.4mm. Others were even more accurate and displayed 2% of failed test cases with a radial distance of 2.02mm to 2.04mm. The set of test cases proved that the system had better accuracy than the requirement, and we felt very satisfied considering the results above.

An interesting finding is that QuickCheck performs some local searching even if not explicitly defined in the test case simplification procedure. In one occasion, QuickCheck checked twice at the same coordinate position, and the system gave two different measurements. Of course, this kind of situation may happen when radio signaling technologies are involved. The important lesson here is that regardless of the fact that two parameters give different results, the system property should be upheld (small differences in the radial distance are disregarded, instead we just make sure that the radial distance is always less than 2mm).

5.5 Planned features for QuickCheck

Throughout this study, we have observed a potential for QuickCheck to support statistical functionalities. Some planned features for future releases include control or specification of number of test cases, and generation of test cases by sampling from a defined set of data (i.e. real patient data). Improving the logging capabilities for QuickCheck could notably expand the potential of using QuickCheck for test results analysis. One possible improvement is to log not only the failing test cases but also the successful test cases.

6. Conclusion

We have described a case study in testing a medical device by using a formal model as basis for automatic generation of test cases with the tool QuickCheck. We found a number of errors in the code we developed and were able to spot inaccuracies in prototype models. Early detection and correction of these errors has lead to a high quality product being developed by the medical company at which the case study was performed. This case study assembles some adequate conditions for using formal models. The model is simple, clear and based on a mathematical formula. Verifying medical devices may not always be like this case, and there may be a need for more complex modeling for the system behavior. Nevertheless we think that the technology is worth trying on more medical equipment.

We also identified some limitations in our case study. One limitation for the testing was that the Auto Setup could only move with a precision of 1mm. This limits the test cases, and does not allow the testing of points between millimeters. In addition, this case study does not cover the necessity of having a given distribution (in this case a normal distribution) and usage of sample data from patients.
When a test fails, we want to obtain the coordinates that give the highest possible measurement fault, in other words, for which the distance to the position is largest.
This is not possible to perform automatically with the current version of QuickCheck. QuickCheck provides simplification of input data, but cannot yet connect it to the result of the actual test.

It is worth mentioning that one of the limitations of working in a lab is the presence of sporadic radio transmission noise due to research activities in nearby companies. This also enforces a sufficient number of tests in order to assure the robustness of the system in less than ideal situations. We can store the test cases sequence in QuickCheck and redo the property execution in order to see any behavior that can be influenced by the environment and signal fluctuations. This would be particularly good if we want to improve the robustness of the system to external noise, which is very common in an environment like a hospital.

The most remarkable aspects of this study focus on several positive results: First, property-based testing proved to be beneficial and feasible within this domain in contrast to the normal tendency of using test suites. QuickCheck provided good support for combining cost-effective regression testing and formal specification. This combination will hold a proof-based certification process for medical devices which are typically systems in continuous evolution. Regarding to this last point, the capacities of QuickCheck for performing high level testing can be regarded as a potential tool that could facilitate the process of integration and system testing within highly complex systems like medical devices.

References

1. J. L. Armstrong, M. Williams, R. Virding, and C. Wikström. ERLANG for Concurrent Programming. Prentice-Hall, 1993.
2. T. Arts, J. Hughes, J. Johansson, and U. Wiger. Testing telecoms software with Quviq QuickCheck. *Proceedings of the 2006 ACM SIGPLAN Workshop*, September 16, 2006.
3. J.W. Brakel. Formal Verification of a Medical Protocol. ISIS Technical Report, University of Twente, 2005.
4. J.P. Bowen and V. Stavridou. Formal methods and software safety. *Safety of Computer Control Systems 1992 (SAFECOMP'92)*, pp. 93–98, Pergamon Press, 1992.
5. J. Bowen and V. Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, volume 8, issue 4, pp. 189–209, 1993.
6. K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. *Proceedings of the Fifth ACM SIGPLAN international Conference on Functional Programming (ICFP '00)*. ACM Press, pp. 268–279, 2000.
7. J. L. Cyrus, J. Daren and P. D. Harry. Formal Specification and Structured Design in Software Development. Hewlett-Packard Journal, 1991.
8. Food and Drug Administration (FDA). General Principles of Software Validation: Final Guidance for Industry and FDA Staff, FDA, 2002.
9. J. Jacky, J. Unger, M. Patrick, D. Reid and R. Risler. Experience with Z developing a control program for a radiation therapy machine. *Proceedings of the 10th International Conference of Z Users*, LNCS, Springer-Verlag, pp. 317–328, 1996.
10. R. Jetley and S. P. Iyer. Enabling Certification through an Integrated Comprehension Approach. *High Confidence Medical Device Software and Systems (HCMDSS) Workshop*, Philadelphia, USA, June 2005.
11. R. Jetley, P. Iyer and P. Jones. A Formal Methods Approach to Medical Device Review *Computer*, volume 39, no. 4, pp. 61–67, 2006.
12. P. Jones. Assurance and Certification of Software Artifacts for High-Confidence Medical Devices. *High Confidence Medical Device Software and Systems (HCMDSS) Workshop*, Philadelphia, USA, June 2005.
13. V. Kasurinen and K. Sere. Integrating action systems and Z in a medical system specification. *FME'96: Industrial Benefit and Advances in Formal Methods*, LNCS 1051, Springer-Verlag, pp. 105–19, 1996.
14. Medical Device Directory (MDD). "Council directive 93/42/EEC of 14 June 1993 concerning medical devices", Medical Device Directory, 2003.
15. M. Marcos, M. Balser, A. ten Teije and F. van Harmelen. From informal knowledge to formal logic: a realistic case study in medical protocols. *Proceedings of 13th Int. Conf. on Knowledge Engineering and Knowledge Management*, LNCS, Springer–Verlag, pp. 49–64, 2002.
16. I. Lee, G. J. Pappas, R. Cleaveland, J. Hatcliff, B. H. Krogh, P. Lee, H. Rubin and L. Sha. High-Confidence Medical Device Software and Systems. *Computer*, volume 39, no. 4, pp. 33–38, 2006.
17. M. Poonawala, S. Subramanian, W. Tsai, R. Mojdehbakhsh and L. Elliott. Testing Safety-Critical Systems - A Reuse-Oriented Approach. *Proceedings of 9th Int. Conf. Software Eng. and Knowledge Eng. (SEKE 97)*, Knowledge Systems Institute, pp. 271–278, 1997.
18. P. Reason and H. Bradbury. Handbook of Action Research. Thousand Oaks, Sage Publications, 2001.
19. J. Rushby. Formal Methods and the Certification of Critical Systems. Computer Science Laboratory, SRI International, Menlo Park, CA. Number SRI-CSL-93-7, December 1993.
20. G. Sharp and N. Kandasamy. A Dependable System Architecture for Safety-Critical Respiratory-Gated Radiation Therapy. *Proceedings of the international Conference on Dependable Systems and Networks (Dsn'06)*, volume 00, DSN, IEEE Computer Society, June 2006.
21. D.R. Wallace, D.R. Kuhn and L.M. Ippolito. An analysis of selected software safety standards. *Proceedings of the Seventh Annual Conference on Computer Assurance (COMPASS '92)*, pp. 123–136, June 1992.
22. A. Benincasa. "Feasibility study for image guided kidney surgery: assessment of required intra-operative surface for accurate physical to image space registrations," M.S. thesis, Vanderbilt University, USA, 2006.
23. D. Avison, F. Lau, M. Myers , P. Nielsen. Action research. Commun ACM 42, pp. 94–97, 1999.
24. T. Gilmore, J. Krantz and R. Ramirez. Action Based Modes of Inquiry and the Host-Researcher Relationship. Consultation 5.3: 161, 1986.
25. S. Kemmis, & R. McTaggart. The Action Research Reader. 3rd edition. Geelong: Deakin University Press, 1988

Glossary

Term	Definition
Coordinates	A set of numbers that identify imaginary points on a reference system. Coordinates describe position in two or three dimensions.
Dose Planning System	Also known as Treatment Planning system, consists of a system which is used for the planning and evaluation of a radiotherapy dosage to treat malignant tumors
Lineal accelerator	A machine that creates high-energy radiation to treat cancers, using electricity to form a stream of fast-moving subatomic particles.
Plate	It consists of a device in the form of a tray where the receivers are placed along with the circuit elements that will send the signal data to the signal processing module of the system.
Receiver	Device for capturing radio frequency signals.
Tolerance margin	It is the maximum offset allowed from the patient isocenter in a given direction in order to assure a safe treatment.
Transmitter	Consists of an electronic device which with the aid of an antenna propagates an electromagnetic signal.
Treatment table	The table that the patient lies on during treatment.
Treatment table coordinate system	Consists of a coordinate system which is stationary with respect to the treatment table.

Appendix: QuickCheck Code used for the Testing

```
-module(position_eqc).

-export([antenna_coordinate_specific/1, upper_corner_coordinate/1,
stringList_to_floatList/1, prop_position/2, antenna_coordinate/0, start_test/0,
stop_test/1]).
-define(PortIn, 5678).
-define(PortOut, 5679).

-compile(export_all).

-include("C:/Program Files/erl5.5.3/lib/eqc-1.07/include/eqc.hrl").
-include("C:/Program Files/erl5.5.3/lib/eqc-1.07/include/eqc_statem.hrl").

start_test()->
       IpAddress = "localhost",
       Portin = 5678,
       Portout = 5679, %Check the port description
       {ok, RSock} = gen_tcp:connect(IpAddress, Portout, [binary, {packet, 0}]),
       ok = gen_tcp:send(RSock, "Connect"),
       {ok, LSock} = gen_tcp:listen(Portin, [binary, {packet, 0}, {active, false}]),
       {ok, Sock} = gen_tcp:accept(LSock),
       {ok, <<"Accept">>} = gen_tcp:recv(Sock, 0),
       gen_tcp:close(Sock),
       {RSock, LSock}.

stop_test({RSock, LSock})->
    {gen_tcp:close(RSock), gen_tcp:close(LSock)}.

prop_position(Margin, {RSock, LSock}) ->
  ?FORALL(Coordinate,antenna_coordinate_borders(),
          eqc:collect(Coordinate,
          begin
             Message = send_coordinates(Coordinate),
             ok = gen_tcp:send(RSock, Message),
             {ok, Sock1} = gen_tcp:accept(LSock),
             {ok, <<"AS_Moved_To_Position">>} = gen_tcp:recv(Sock1, 0),
               ok = gen_tcp:send(RSock, "Request_Answer"),
             {ok, Sock2} = gen_tcp:accept(LSock),
             {ok, Bin} = gen_tcp:recv(Sock2, 0),
             Position = binary_to_list(Bin),
             inside_margin(Position, Coordinate, Margin)
          end)).


prop_find_working_margin (Margin, {RSock, LSock}) ->
  ?FORALL(Coordinate,antenna_coordinate(),
          begin
               Message = send_coordinates(Coordinate),
               ok = gen_tcp:send(RSock, Message),
               {ok, Sock1} = gen_tcp:accept(LSock),
               {ok, <<"AS_Moved_To_Position">>} = gen_tcp:recv(Sock1, 0),
               ok = gen_tcp:send(RSock, "Request_Answer"),
               {ok, Sock2} = gen_tcp:accept(LSock),
               {ok, Bin} = gen_tcp:recv(Sock2, 0),
               Position = binary_to_list(Bin),
              not inside_margin(Position, Coordinate, Margin)
          end).

extrapolated_coordinate(Coordinate)->
    X = 135 - abs(lists:nth(1,Coordinate)-35),
    Y = 140 - abs(lists:nth(2,Coordinate)-40),
    Z = 172 - abs(lists:nth(3,Coordinate)-72),
    [X, Y, Z, lists:nth(4,Coordinate), lists:nth(5,Coordinate)].
```

```
prop_symmetric(Margin)->
  ?FORALL(Coordinate,antenna_coordinate (),
          begin
              Extrapolated = extrapolated_coordinate(Coordinate),
              Message = send_coordinates(Coordinate),
              ok = gen_tcp:send(RSock, Message),
              {ok, Sock1} = gen_tcp:accept(LSock),
              {ok, <<"AS_Moved_To_Position">>} = gen_tcp:recv(Sock1, 0),
              ok = gen_tcp:send(RSock, "Request_Answer"),
              {ok, Sock2} = gen_tcp:accept(LSock),
              {ok, Bin} = gen_tcp:recv(Sock2, 0),
              Position = binary_to_list(Bin),

              Message2 = send_coordinates(Extrapolated),
              ok = gen_tcp:send(RSock, Message2),
              {ok, Sock11} = gen_tcp:accept(LSock),
              {ok, <<"AS_Moved_To_Position">>} = gen_tcp:recv(Sock11, 0),
              ok = gen_tcp:send(RSock, "Request_Answer"),
              {ok, Sock22} = gen_tcp:accept(LSock),
              {ok, Bin2} = gen_tcp:recv(Sock22, 0),
              Position2 = binary_to_list(Bin2),
              Distance1 = radial_distance(Position, Coordinate),
              Distance2 = radial_distance(Position2, Extrapolated),
              abs(Distance1 – Distance2) = < 1
          end).

send_coordinates(Coordinates) ->
    "Send_Coordinates:"++lists:map(fun(X) -> integer_to_list(X)++"," end, Coordinates).

inside_margin(Position, Coordinate, Margin)->
    Distance = radial_distance(Position, Coordinate),
    Distance =< Margin.

radial_distance(Position, Coordinate)->
    Position_List = stringList_to_floatList(Position),
    Distance = math:sqrt(math:pow((lists:nth(1,Position_List)-lists:nth(1,Coordinate)),
2)+ math:pow((lists:nth(2,Position_List)-lists:nth(2,Coordinate)), 2)+
math:pow((lists:nth(3,Position_List)-lists:nth(3,Coordinate)), 2)), Distance.


antenna_coordinate() ->
   [choose(35,135),choose(40,140),choose(72, 172), choose(-20, 20), choose(-20, 20)].
```

Acknowledgements

We would like to thank our Supervisor Thomas Arts for his valuable coaching through the project. Your advice and support made this project a unique and rich experience.

Special thanks to Tomas Gustafsson, CEO of Micropos, Johan Linder for his coaching, and all the staff at Micropos for their support. Thanks to you it was possible to perform this study.

To the Ministry of Science and Technology of Costa Rica for supporting the studies of Aiko Fallas Yamashita through their Science and Technology incentive program

We would also like to thank our ex-Program Manager Thomas Lundqvist, for his feedback and critical spirit towards our work.