Open-Source Software & Personal Medical Devices: Interpreting Risk Through an Evaluation of Software Testing

Michael Dorin / Heather Mortensen / Sergio Montenegro

The availability of powerful low-cost hardware and advanced software tools has made open-source medical devices possible. Deciding to use an open-source medical device may require acceptance of some risk. Fully comprehending the risk level is essential since failure of the software or the medical device is dangerous. As many medical applications contain complicated codes, an excellent method for understanding software readiness is to evaluate how much testing has been completed on the software. As a case study, this project evaluates the level of testing performed on software components of the Loop Artificial Pancreas system.

Classic methods for evaluating complicated source codes are used to demonstrate how much testing is needed in a project. Our analysis shows that the Loop Artificial Pancreas system (master branch) has been thoroughly tested with most of the faults likely discovered. By using classic software engineering metrics and techniques, it is possible to gauge how completely an open-source medical product has been tested and make an educated decision about the risk associated with using it.

Software de código abierto y dispositivos médicos personales: interpretación del riesgo a través de una evolución de las pruebas de *software*

La disponibilidad de *hardware* de bajo costo y herramientas de *software* avanzadas ha hecho posible los dispositivos médicos de código abierto. La decisión de utilizar un dispositivo médico de código abierto puede requerir la aceptación de algún riesgo. Comprender completamente el nivel de riesgo es esencial ya que la falla del *software* o del dispositivo médico es peligrosa. Como muchas aplicaciones médicas contienen códigos complicados, un método excelente para comprender la preparación del *software* es evaluar la cantidad de pruebas que se han completado en el *software*. Como estudio de caso, este proyecto evalúa el nivel de pruebas realizadas en componentes de *software* del páncreas artificial de bucle.

Los métodos clásicos para evaluar el código fuente complicado se utilizan para demostrar cuántas pruebas se necesitan en un proyecto. Nuestro análisis muestra que el sistema de páncreas artificial de bucle (rama maestra) se ha probado exhaustivamente con la mayoría de las fallas probablemente descubiertas. Mediante el uso de técnicas y métricas de ingeniería de *software* clásico, es posible medir cuán completamente se ha probado un producto médico de código abierto y tomar una decisión informada sobre el riesgo asociado con su uso.

Open Source Software & Personal Medical Devices: Interpreting risk through an evaluation of software testing

Michael Dorin, Heather Mortensen, Sergio Montenegro

mike.dorin@stthomas.edu, mort0048@stthomas.edu, sergio.montenegro@uni-wuerzburg.de

ABSTRACT

The availability of powerful low-cost hardware and advanced software tools has made open-source medical devices possible. Deciding to use an open-source medical device may require acceptance of some risk. Fully comprehending the risk level is essential since failure of the software or the medical device is dangerous. As many medical applications contain complicated codes, an excellent method for understanding software readiness is to evaluate how much testing has been completed on the software. As a case study, this project evaluates the level of testing performed on software components of the Loop Artificial Pancreas system. Classic methods for evaluating complicated source codes are used to demonstrate how much testing is needed in a project. Our analysis shows that the Loop Artificial Pancreas system (master branch) has been thoroughly tested with most of the faults likely discovered. By using classic software engineering metrics and techniques, it is possible to gauge how completely an open-source medical product has been tested and make an educated decision about the risk associated with using it.

CASE STUDY

LOOP ARTIFICIAL PANCREAS

Loop was funded and built primarily by diabetic patients and caregivers. It was the first closed-loop system available in the US and preceded the release of a commercial system. Legal liability was limited by the fact that individual patients built the system themselves atop existing commercial medical devices.

Loop replaces an insulin pump controller and a continuous glucose monitor (CGM) receiver with an iPhone. A CGM sensor transmits blood glucose measurements via Bluetooth low energy (BLE) to the iPhone. The iPhone performs analytics and predictive statistics. Data is output to the user, alongside alerts for impending low blood sugar levels. Loop can utilize one of four different insulin models to perform the analysis. Loop incorporates data manually collected by the user inside Apple's Health app. In the event of Loop failure, insulin delivery reverts to default delivery rates in the pump. The iPhone sends BLE to the RileyLink, a custom-built piece of hardware that bridges communications between devices that use BLE (iPhone) and devices that use radio frequency (RF) (insulin pumps). RileyLink sends an RF command (916 MHz) to an insulin pump, where physical delivery of the medication to the user takes place through a catheter. [5] [6] [9]

METHODOLOGY

1. Estimate potential bugs using Halstead method 1

   Bugs = (Halstead effort$^{2/3}$)/3000. [2]
2. Estimate potential bugs using Halstead method 2

   Bugs = Halstead volume/3000 [2]
3. Estimate potential bugs using McConnell's method

   A delivered bug every 15 to 50 lines of code. [4]
4. Akiyama's method

   Bugs = 4.86+0.018 X (lines of code) [1]
5. Estimate reported bugs reviewing online reporting.
6. Estimate reported bug count by scanning git log.
7. Compare estimated bugs to reported bugs.

IMPORTANT ONGOING WORK

Loop is a mature project, so it was possible to evaluate it at a very tested state. However, new open-source medical and aerospace projects are released on a regular basis. More work is required to design a practical procedure using commonly available and affordable tools and techniques for evaluating risk on mission critical projects. The following aspects are being explored for the future procedure.

McCabe's cyclomatic complexity gives an indication of how many independent paths exist in a module by showing how many unit tests are required. Unit tests should be reviewed and execution reports analyzed [3].

Static analysis is a procedure for finding program faults using tools which examine the code without executing it.

Human Complexity Analysis measures the difficulties a person may encounter when reviewing the source code. The coding style is an important part of this.

Weibull Analysis will make a prediction based on bug discovery rates when about 2/3 of bugs are found. Weibull will be especially useful on new projects. [7]

Table 1

Project	Online Reported Bugs	Bugs Keyword in Log
Amplitude-iOS	70	176
CGMBLEKit	28	52
G4ShareSpy	0	4
LoopKit	35	188
SwiftCharts	311	169
dexcom-share-client-swift	3	5
Rileylink iOs	67	347
Loop	458	415
Total	**972**	**1356**

Table 2

Estimated Bugs	Technique	Online Reported	Keywords Reported
1284.156	Akiyama's Method	75.6 %	100 %+
1066.08	McConnell's Method	91.1 %	100 %+
936.16	Halstead 1	96.3 %	100 %+
1388.04	Halstead 2	70 %	97.6 %

DISCUSSION AND CONCLUSION

The number of bugs found in a project is a good representation of how much progress has been made in the test workflow. With open-source projects, we found that not all bugs are reported to the official bug tracking software. Often developers make a fix and check it in without officially logging the bug. Also many times users of a particular project report bugs that do not inspire a fix from the developers. In this research, a scan of the git log as well as a review of on-line reported bugs were used to estimate bug fixes.

The particular tools and metrics used in this project were selected and evaluated because of their availability and ease of understanding for those with only a basic background in software development. It is imperative that a person considering the use of one of these devices has a practical means of evaluating their risk, without the requirement of complicated or expensive tools. All of the tools used in this project are open source.

This project and case study evaluated the state of software testing for a particular project. Software testing alone is essential, but it does not address all the areas of concern. Further research is necessary to build a complete set of practical tools and procedures. This project and case study have shown one important and practical method for evaluating open-source mission-critical systems.

REFERENCES

1. Akiyama, F. , An example of software debugging. IFIP Congress. (1971)
2. Halstead. Elements of software science. Vol. 7. New York: Elsevier, 1977.
3. McCabe, "A complexity measure." IEEE Transactions on software Engineering 4 (1976)
4. McConnel, "Code Complete" Pearson Education (2004).
5. DiSimone, K, "My Artificial Pancreas", https://myartificialpancreas.net/, 2017
6. LoopDocs, https://loopkit.github.io/loopdocs/#welcome-to-loop, Retrieved July 2019.
7. Simmons, Erik. "Software Defect Arrival Modeling Using the Weibull Distribution." Pacific Northwest Software Quality Conference.
8. Pixabay, pixabay.com. Pictures retrieved July 2019
9. MT pump frequency: https://loopkit.github.io/loopdocs/faqs/rileylink-faqs/

ACKNOWLEDGEMENTS

We wish to especially thank Kathryn DiSimone and Nate Racklyeft for their assistance with information and images, as well as necessary permissions.