A Framework for Software Health Management using Bayesian Statistics —Position Paper—

Yuning He
yuning.he@nasa.gov
NASA Ames Research Center
Moffett Field, CA

Johann Schumann
johann.m.schumann@nasa.gov
SGT, Inc/KBR, NASA Ames Research Center
Moffett Field, CA

ABSTRACT

As size and complexity of safety-critical software systems increase, Software Health Management (SWHM) must make sure that the software always remains in safe and healthy regions of the state space. Boundaries between healthy and unhealthy regions are important for the detection of violations and health management. In this position paper, we present a framework, which employs techniques from Bayesian statistical modeling and active learning to efficiently characterize health boundaries in high-dimensional spaces. We will discuss, how this framework supports SWHM during design time and during operation of learning/adapting software systems.

ACM Reference Format:
Yuning He and Johann Schumann. 2020. A Framework for Software Health Management using Bayesian Statistics —Position Paper—. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3387940.3392208

1 INTRODUCTION

Size and complexity of software found in safety-critical systems has increased tremendously. Systems like self-driving automobiles, unmanned aerial systems (UAS), or large and complex IoT (Internet of Things) systems need large software components for proper operation, which often contain machine-learning a lgorithms, artificial intelligence (AI) systems, or advanced and complex sensor and image processing systems. Their proper operation, under all circumstances, is of utmost importance. Malfunctions of such systems can cause loss of human life, substantial damage, or mission failure.

Therefore, such systems are considered highly safety-critical and rigorous verification and validation (V&V) of the software is mandatory. However, most traditional V&V techniques cannot be applied to such advanced systems. For example, the V&V of Deep Neural Networks (DNN) is still in its infancy [3], although numerous techniques have been developed.

Here, techniques for Software Health Management (SWHM) [16], are important and useful to ensure that the software and system always remains safe, even under adverse conditions. Typically, runtime monitors are the basis of Health Management Systems that, combined with a reasoning component [15], can detect if the software is healthy or not while the system is in operation. A deviation of system safety, performance, or other behavior from its nominal behavior can not only point to a specific failure, but also provides information about a decline in system and software health. In such a case, contingency measures, like reconfiguration, use of alternative software, or graceful degradation, just to mention a few, can be triggered to attempt to return the software back into a healthy state, or at least try to minimize damage. Numerous successful applications and approaches toward SWHM, often combined with diagnostics and prognostics components have demonstrated the necessity of SWHM for large and safety-critical real-life software applications [16].

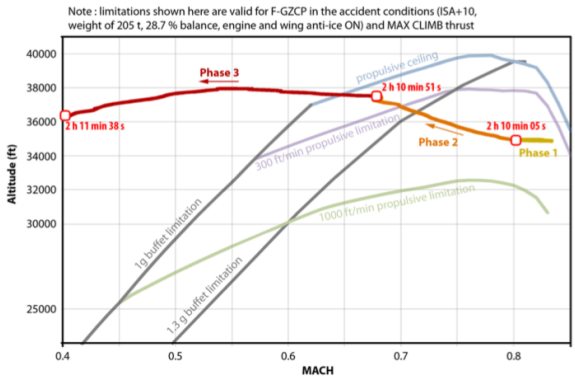

Figure 1: Trajectories of the flight AF-447. Image from [2]

A system obviously can only operate safely within given boundaries. Such safety-boundaries, exist in a usually high-dimensional state space of the system and separate safe system/software operation from unsafe behavior. A simple example might be the operational characteristics of an aircraft [9]: the range of airspeed, under which an aircraft can safely stay in the air strongly, and non-linearly depends on parameters like temperature and altitude. The pilot (or the controlling autopilot software) must always ensure that the aircraft remains within this safety envelope. Only then, system and software health can be assured. This situation is demonstrated in Figure 1, which shows the speed (in % of speed of the sound) and altitude of the ill-fated flight AF-447 before its crash into the Atlantic Ocean [2]: the solid line annotated with time points marks the behavior of the aircraft. Starting out with a good speed of about

ICSEW'20, October 5–11, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3392208

0.83 mach at an altitude of 35,000ft, sensor failures (in the Pitot tubes) and a resulting erroneous behavior of the flight software caused a reduction of speed of the aircraft at roughly the same altitude (point of operation moves to the left on the solid line). Whereas the area on the lower right is a safe operational region for the aircraft, lower speeds at high altitude cause stalls and lead to a crash. Several safety-boundaries (inverted U-shaped curves) that have been determined by the aircraft designer separates these areas. Due to the failures and misinterpretation of information by the pilots, the aircraft leaves the safe operational area and enters unsafe regions, eventually leading to the fatal crash; the system state thus goes from healthy to extremely healthy.

SWHM must be aware of such safety boundaries in order to be able to determine the health of the system and software. As becomes evident from Figure 1, such boundaries are often non-linear and can have complicated geometric shapes. Figure 2 shows an abstract high-level view for a two-dimensional projection, which contains several regions of good health. SWHM must ensure that the operating point of the system remains well inside the health and safe regions at all times. In Figure 2, the blue operating point is well in healthy region; the magenta circle, however, is close to the boundary, indicating a potential health problem. It is obvious that the SWHM has to be aware of the safety and health boundaries.

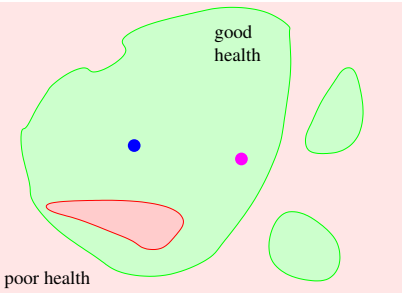

Figure 2: Healthy and unhealthy regions in 2D state space. Health boundaries are shown as solid lines.

In this position paper, we present a statistical framework for the detection and characterization of safety and health boundaries. A surrogate model of the system is constructed by exercising the system with generated test cases or scenarios. The results of each test are used to incrementally improve the statistical surrogate model. By using methods from computer experiment design and active learning, the number of cases to reliably model the areas near the boundaries can be kept low. We present the architecture of our framework, which can also estimate the boundary shapes and return easy-to-understand geometric descriptions that can provide feedback to the system designer or software health management specialist.

Our framework is mainly being used for the analysis and SWHM of software systems that interact with the physical world (cyber-physical systems). Although those systems are controlled by software, the health boundaries are often defined or at least influenced by physical laws (e.g., aircraft dynamics) and thus tend to be continuous.

Software per se, on the other hand is discrete. However, when considering larger and complex software systems, their behavior can often be approximated by continuous models and thus amenable to analysis by our framework. Typical examples, as might show up, for example, in distributed systems or Linux software ecosystems, include use of resources (memory, CPU load), response times, failure rates, and such. Allowing such a system to enter unsafe and unhealthy regions can lead to overall degradation of system health and potentially catastrophic results. For example, the Mars rover "Spirit" [13], went into a reboot loop, because the on-board file system had not been emptied prior to landing and was thus growing too large. That growth, however, had not been monitored.

Therefore, our framework can be applied to complex and potentially distributed software systems as, for example, in software eco-systems [1] Here, software health is often related to performance criteria, e.g., resource consumption, response times, availability of services. Here again, boundaries can be found that separate the behavior of a healthy system from behavior caused by an unhealthy software. Our framework can also be applied to systems, that can change dynamically due to unforeseen circumstances, failures, reacts to environmental influences or undergoes evolutionary processes. A very important question for SWHM for such system is on how to ensure that the health of the software system can be retained during such an adaptation and change.

Techniques, which are being used successfully for the prognosis of the health state of physical systems, like batteries [6] or power electronics [18], or hydraulic valves [11] could also be easily adapted toward prognostics algorithms for software systems.

In the remainder of this position paper, we briefly describe our framework based upon hierarchical statistical modeling and techniques of Computer Experiment Design and active learning, and then discuss the application of out framework for Software Health Management of large and complex safety-critical systems.

2 SWHM ARCHITECTURE

Figure 3 shows the high-level architecture of our framework for the analysis of system and software health. On the right-hand side of the figure, we have our "system under test", the software system, for which health boundaries must be determined and characterized. The system is exercised using a newly generated test case or scenario, which has been produced by our framework as described below. The system behavior is evaluated and compared against given health requirements in order to determine if the system under test is healthy or not under the given conditions and parameters.

The result of the test run is then used to refine our statistical health surrogate model (Figure 3-left). For the representation and construction of the model, we are using Dynamic Regression Trees (DynaTrees) [7, 17], a dynamic Gaussian process model based upon Particle Filters. DynaTrees are regression and classification learning models with complicated response surfaces in on-line application settings. This kind of models is thus ideally suited to capture and represent the boundaries that separate healthy from unhealthy system states.

DynaTrees create a sequential tree model whose state changes over time with the accumulation of new data. We use this property

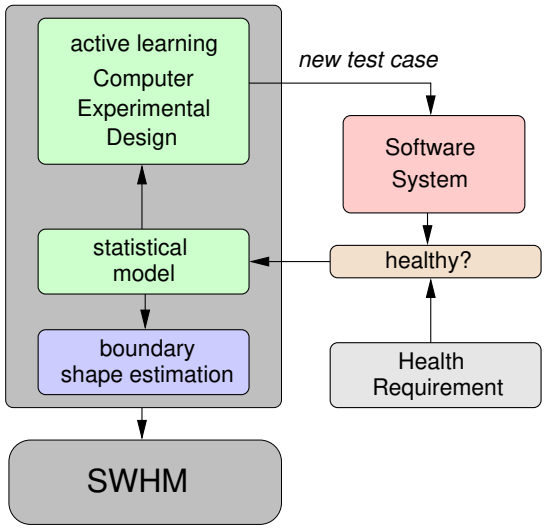

Figure 3: High level architecture of our testing framework

to incrementally refine the model using new test cases that are produced by our active learning module (left top of Figure 3).

For the efficient generation of most informative test cases, we use active learning and techniques from Computer Experimental Design. Classical active learning algorithms (e.g., [5, 12]) use search metrics that focus on under-explored regions in the domain space in order to substantially reduce the number of test cases to characterize the model, compared to combinatorial exploration or randomized methods. Inspired by [10] and work on contour finding algorithms, we loosely follow [14] and define our boundary-aware metric boundary-EI (Expected Improvement) [8, 9] that puts the focus of the search into regions, which are close to boundaries, which separate safe from unsafe regions. Because our framework constructs a model of the response probability surface (with the boundary defined by $p = 0.5$), our algorithm can deal with continuous health assessments in the range between 0 and 1.

Using our EI metric, our framework explores the input space in an intelligent manner, focusing on finding new data points in "interesting" and potentially "troublesome" areas, near boundaries between healthy and unhealthy regions. This exploration able to cover the entire input space with a low number of data points.

Our framework can also estimate and characterize the geometric shapes of the boundaries using Bayesian techniques [8]. These parameterized shapes (e.g., hyperplanes, spheres, polygons), taken from a library can provide valuable feedback and insights to the designer of the software system, can facilitate the understanding of the current system health and support decision making.

3 DISCUSSION

Many traditional systems for software health management are monitoring the system health against a number of predefined "thresholds". In contrast, our framework uses efficient techniques to iteratively construct a statistical surrogate model for representing the health boundaries that separate health regions of the software from unhealthy ones. Such boundaries in the high-dimensional state space can have complex and nonlinear forms that (a) would usually not be represented by design-time "thresholds", and (b) reflect the real behavior of the system in its current state and environment.

The latter is of particular importance, if the software changes or adapts over time and does not necessarily behave exactly as originally designed. An adaptive aircraft controller, like the NASA Intelligent Flight Control System (IFCS) [4] adapts toward a damage to keep the aircraft safe and controllable in the air. Software systems, which contain machine-learning or AI components or feature Self-* properties [19] also belong into this category. Similarly, growing software eco-systems may exhibit changing, adaptive, or emerging behavior. Health management for a software system that changes over time to, for example, adapt toward a (hardware) failure or damage, or uses online technique to dynamically optimize its behavior, poses additional challenges: Firstly, the temporal aspects of the adaptation must be considered for health management. For example, if the adaptation of a damage-adaptive controller happens too fast, oscillations may occur, which could lead to dangerous situations and poor/non-existing system health. Secondly, transients that can occur during system adaptation or reconfiguration, are usually a sign of poor software health if they become too large. Finally, the health boundaries of the software can change in potentially unforeseen ways during adaptation.

In such situations, on-line determination of the health-boundaries, which change dynamically during system operation is of utmost importance. Figure 4 shows how health-boundaries can change during system adaptation, learning, or optimization. Restriction to statically defined boundaries could lead to wrong health assessments.

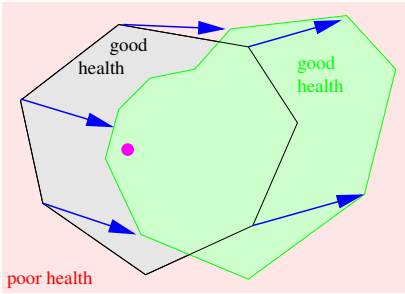

Figure 4: Software Health boundaries are changing during adaptation/learning/optimization (blue arrows). Operating point (magenta) is well in healthy region before adaptation, but close to unhealthy area after adaptation.

Such changing boundaries must be characterized during system operation. Therefore, the effort necessary to determine these boundaries must be kept as low as possible. Our customized metrics for active learning can the number of required test cases and system evaluations in high-dimensional spaces.

We are therefore confident that our statistical framework using hierarchical models and active learning can provide substantial benefits for system and software health management, in particular for adaptive and learning systems, like AI systems.

REFERENCES

[1] Carina Alves, Joyce Aline Pereira de Oliveira, and Slinger Jansen. 2017. Software Ecosystems Governance - A Systematic Literature Review and Research Agenda. In *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 3, Porto, Portugal, April 26-29, 2017*, Slimane Hammoudi, Michal Smialek, Olivier Camp, and Joaquim Filipe (Eds.). SciTePress, 215–226. https://doi.org/10.5220/0006269402150226

[2] BEA. 2012. *Final Report on the accident on 1st June 2009 to the Airbus A330-203 operated by Air France Flight AF 447.* Technical Report. Bureau d'Enquêtes et d'Analyses pour la sécurité de l'aviation civile.

[3] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. 2018. Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. *CoRR* abs/1812.05389 (2018). arXiv:1812.05389 http://arxiv.org/abs/1812.05389

[4] John Bosworth and Peggy Williams-Hayes. 2007. *Flight Test Results from the NF-15B Intelligent Flight Control System (IFCS) Project with Adaptation to a Simulated Stabilator Failure.* Technical Report NASA/TM-2007-214629. NASA.

[5] D. A. Cohn. 1996. Neural Network Exploration using Optimal Experimental Design. *Advances in Neural Information Processing Systems* 6, 9 (1996), 679–686.

[6] M. Daigle and C. Kulkarni. 2013. Electrochemistry-based Battery Modeling for Prognostics. In *Annual Conference of the Prognostics and Health Management Society 2013*. 249–261.

[7] R. Gramacy and N. Polson. 2011. Particle learning of Gaussian process models for sequential design and optimization. *Journal of Computational and Graphical Statistics* 20, 1 (2011), 467–478.

[8] Yuning He. 2012. *Variable-length Functional Output Prediction and Boundary Detection for an Adaptive Flight Control Simulator.* Ph.D. Dissertation. University of California at Santa Cruz.

[9] Yuning He. 2015. Online detection and modeling of safety boundaries for aerospace applications using active learning and Bayesian statistics. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*. IEEE, 1–8. https://doi.org/10.1109/IJCNN.2015.7280595

[10] D. Jones, M. Schonlau, and W. J. Welch. 1998. Efficient Global Optimization of Expensive Black Box Functions. *Journal of Global Optimization* 13 (1998), 455–492.

[11] C. Kulkarni, M. Daigle, G. Gorospe, and G. Goebel. 2014. Validation of Model-Based Prognostics for Pneumatic Valves in a Demonstration Testbed. In *Annual Conference of the Prognostics and Health Management Society 2014.*

[12] D. J. C. MacKay. 1992. Information–based Objective Functions for Active Data Selection. *Neural Computation* 4, 4 (1992), 589–603.

[13] T. Neilson. 2005. The Mars Rover Spirit FLASH anomaly. 4186 – 4199. https://doi.org/10.1109/AERO.2005.1559723

[14] Pritam Ranjan, Derek Bingham, and George Michailidis. 2008. Sequential Experiment Design for Contour Estimation from Complex Computer Codes. *Technometrics* 50, 4 (2008), 527–541.

[15] J. Schumann, T. Mbaya, O. Mengshoel, K. Pipatsrisawat, A. Srivastava, A. Choi, and A. Darwiche. 2013. Software Health Management with Bayesian Networks. *Innovations in Systems and Software Engineering* (2013).

[16] Ashok N Srivastava and Johann Schumann. 2013. Software health management: a necessity for safety critical systems. *Innovations in Systems and Software Engineering* 9, 4 (2013), 219–233.

[17] Matthew A. Taddy, Robert B. Gramacy, and Nicholas G. Polson. 2011. Dynamic Trees for Learning and Design. *J. Amer. Statist. Assoc.* 106, 493 (2011), 109–123. http://EconPapers.repec.org/RePEc:bes:jnlasa:v:106:i:493:y:2011:p:109-123

[18] Lifeng Wu, Jing Yang, Zhen Peng, and Hongmin Wang. 2017. Remaining useful life prognostic of power metal oxide semiconductor field effect transistor based on improved particle filter algorithm. *Advances in Mechanical Engineering* 9, 12 (2017), 1687814017749324. https://doi.org/10.1177/1687814017749324 arXiv:https://doi.org/10.1177/1687814017749324

[19] P. Zhou, D. Zuo, K. M. Hou, Z. Zhang, J. Dong, J. Li, and H. Zhou. 2019. A Comprehensive Technological Survey on the Dependable Self-Management CPS: From Self-Adaptive Architecture to Self-Management Strategies. *Sensors (Basel, Switzerland)* 19, 5 (2019), 1033. https://doi.org/10.3390/s19051033