Human in the Loop: What is the Point of no Return?

Rogério de Lemos
University of Kent, UK
r.delemos@kent.ac.uk

ABSTRACT

The main goal of any feedback control system is essentially to remove humans from the loop. This has always been the goal in the engineering of control systems. The MAPE-K loop is the embodiment of a feedback control loop in self-adaptive software systems, but the complete removal of humans from the control loop has not been thoroughly debated. One of the reasons is that, software systems are social-technical systems, and as such, humans need to be considered right from the inception of such systems, otherwise their deployment is bound to fail. However, as software self-adaptation progresses, enabling to place higher assurances on the deployment of these systems to the point humans become dispensable, some ethical questions need to be raised. Similar questions have been raised in past when the first automatic systems became intrinsic to the industrial fabric. The difference between then and now is that then the impact was confined to portions of the society, but now the implications are much wider, if we consider, in particular, software systems that are able to change themselves. If humans are not aware of those changes, and their implications, humans cease to be in tune with the system they are operating, and inevitably accidents will ensue. The point of no return in self-adaptive software systems refers to the moment in their technical maturity when any human involvement with the operation of a system is perceived to create more harm than benefit. Confronted with this situation, software engineers need start asking themselves some basic ethical questions. Do we really need to consider humans as an integral part of self-adaptive software systems? If humans are removed from the control loop, what kind of assurances will be needed for society to accept such systems?

CCS CONCEPTS

• **Software and its engineering → Abstraction, modeling and modularity**; **Software evolution**; **Software verification and validation**; **Software design tradeoffs**.

KEYWORDS

self-adaptive software systems, human in the loop, MAPE-K, resilience, software configuration, assurances

ACM Reference Format:
Rogério de Lemos. 2020. Human in the Loop: What is the Point of no Return?. In *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3387939.3391597

SEAMS '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7962-5/20/05…$15.00
https://doi.org/10.1145/3387939.3391597

1 MOTIVATION

The point of no return (PONR) in aeronautical terms refers to the point on a flight at which a plane is no longer capable to returning to the point of departure. In general terms, PONR refers to the critical point at which turning back or reversal is not possible. Examples of PONR are: setting off an explosion, launching a rocket, or parachuting from an aircraft. In this paper, we refer to the PONR as the moment in the technical maturity of self-adaptive software systems when any human involvement with the operation of a system is perceived to create more harm than benefit. For example, in an autonomous vehicle, if a driver interferes with a critical manoeuvre by braking when faced with a perceived dangerous situation, s/he may cause a serious accident. This might have happened because of the inability of the driver to maintain an accurate view of the actual operational state of the vehicle.

The operational state of an autonomous vehicle, however, should not be restricted to the vehicle itself and to the immediate vicinity of the vehicle, for example, other vehicles, people, obstacles or hazards in general. It should also refer to the state of the software system managing the autonomous vehicle. As in any feedback control system, a stimulus may have different consequences depending on the system state. In the case of autonomous vehicles, depending on the architectural configuration of the software, for example, an action by the driver may lead to different outcomes. Whether an autonomous vehicle can be sophisticated enough to be able to handle potential discrepancies between the actual operational state of the vehicle and the state being perceived by the driver is huge challenge since what the computational system observes may be quite different from what a human driver is able to observe. Moreover, any action from the human driver is based on intent, and the gap between driver's intent and a potential reaction from the autonomous vehicle is huge since the latter is dependent on the operational state of the software system, and the driver is not expected to have access to this. Even if both vehicle and driver are able to observe the same phenomena, the actions taken by the autonomous vehicle and the human driver may be disparate because of the differences in their respective decision making processes. Hence, the conflict between humans and machines. This is not new. The difference now is that humans may not have any influence whatsoever on the decisions to be taken by machines. Using a filming analogy, humans will not be in any supporting role, they will be mere extras.

In this position paper, we will be looking into the impact that the operational state of software, that is supported by a feedback control loop, might have upon the decisions to be taken by the software and the users of the system in which the software is embedded [2]. The

focus will be on self-adaptive software systems, which are a class of systems that are able to manage themselves. In particular, the impact that structural changes [1], related to architectural configurations of software, might have upon the human perception of the operational state of the system in which the software is embedded. Self-adaptive software systems will be an integral part of future autonomous systems, and that is the reason for software engineers to start questioning their role when engineering these type of systems.

2 CONFIGURATION AWARENESS

In order to stimulate the discussion regarding the impact of configurations upon the resilience of systems, in the following, we look into some accidents from the aeronautical industry, one of the safest industries. But first, lets analyse the role of fully trained pilot when faced with an automated system.

John Rushby analysis on the interaction of pilots with cockpit interfaces has identified mental models as a fundamental guide for pilots to interact with automated systems. These mental models are sourced by the instruments on the cockpit as a result of rigorous training supported by operational manuals [5]. The instruments on the cockpit interface are there to provide a perspective of reality that is captured through sensors and gauges. Any discrepancy between reality and instrumentation, or instrumentation and mental models may lead to hazardous situations.

In the following, we succinctly describe some aeronautical accidents that clear capture the role of cockpit instrumentation and pilot mental models. The crash of Air France Flight 447 [6] was blamed on the pilot because he had lost awareness of the operational state of the aircraft. The information provided by the instruments was confusing because one of the sensors, a pitot tube that reads the velocity of the plane, produced erroneous readings, thus affecting the auto-pilot. The pilot relative low experience, compounded by a fully automated cockpit, cause the Airbus 330 to crash into the Atlantic Ocean on a schedule flight between Rio de Janeiro and Paris. This is a typical case in which an instrument failure can be characterised as the initiating event in an accident sequence, however, other identical aircraft had suffered similar pitot tube failures, but pilots were able to circumvent them. A distinct aspect of this accident was that the pilot had other sources of information in the cockpit for obtaining the operational state of the aircraft, but these were ignored. The pilot's mental model of the aircraft operational state was so inaccurate that all his measures to recover the state of the aircraft were inconsistent with what the instruments were showing.

The other two accidents are related to Boeing 737 Max [3]. Both Lion Air Flight 610 and Ethiopian Airlines Flight 302 accidents were caused by a malfunction in the flight control system component, called MCAS (Manoeuvring Characteristics Augmentation System). The role of MCAS was to push the nose of the plane down when it perceived the plane to be exceeding its angle of attack limits, which may lead to an aerodynamic stall. The existence of this component was unknown to the pilots. No training was provided to the pilots on how to fly the aircraft when MCAS was activated. So when MCAS started to interfere with the fight, the pilots could not understand the behaviour of the aircraft, and all their actions to recover the flight path were futile simply because they were dealing with an unknown configuration of the aircraft. It has been recognised that these accidents were no fault of the pilots because none of the training the pilots had received or manuals available on the cockpit were of no avail.

The last accident case clearly shows the need for configuration awareness of the system by their human operators. The same configuration awareness is crucial when humans interact with self-adaptive software systems - if there is the need to do so. In other words, the rationale behind the decisions that a system might take by itself should be explainable to a human. However, complications may arise. For example, let us assume that a self-adaptive software system, which is a component of an autonomous systems, is able to adapt its architectural configuration through several distinct adaptations. If there is a situation in which that software system is not able anymore to take care of itself and raises an exception to a human operator, in other words, it has reached a boundary condition of failure, what actions is a human administrator able to take if the state of the system has evolved to the point that is impossible to establish the trajectory of its operational state? Hence the point of no return, depending on the degree of "selfness" (not the degree of adaptability) that a self-adaptive software system might have, the operational state of the system might become sufficiently obscure for any human involvement.

3 WHAT THE FUTURE WILL BE ASKING US?

Going back to the motivation example, some key questions that may be raised. What will be needed in terms of assurances that will provide the confidence for removing all forms of instrumentation and control from autonomous vehicles, including steering wheels and pedals? [1] In a scenario in which all the vehicles are autonomous, what role should be given to drivers that may have different degrees of training and awareness while interacting with an autonomous vehicle? The point of no return will happen when the risk of empowering the driver with the controls of their vehicles will excel the risk of completely removing the human from the control loop [4].

In terms of self-adaptive software systems, what assurances need to be provided (to certification bodies) by software engineers that their self-adaptive software system is safe and secure (or resilient) to be an integral part of an autonomous system.

REFERENCES

[1] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. 2009. Modeling Dimensions of Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems*. Springer, 27–47.

[2] D. Dörner. 1997. *The Logic of Failure: Recognizing and Avoiding Error in Complex Situations*. Addison-Wesley Pub.

[3] Jack Nicas, Natalie Kitroeff, David Gelles, and James Glanz. 2019. Boeing Built Deadly Assumptions Into 737 Max, Blind to a Late Design Change. *The New York Times* (2019). https://www.nytimes.com/2019/06/01/business/boeing-737-max-crash.html [accessed: 23.01.2020].

[4] Ashley Nunes, Bryan Reimer, and Joseph Coughlin. 2018. People must retain control of autonomous vehicles. *Nature* 556 (04 2018), 169–171. https://doi.org/10.1038/d41586-018-04158-5

[5] John Rushby. 2001. Modeling the Human in Human Factors. In *Computer Safety, Reliability and Security*, Udo Voges (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 86–91.

[6] Jeff Wise. 2011. What Really Happened Aboard Air France 447. *Popular Mechanics* (2011). https://www.popularmechanics.com/flight/a3115/what-really-happened-aboard-air-france-447-6611877/ [accessed: 23.01.2020].

[1] Voice commands might be available, but that is different discussion.