**Sujets de thèse 2017**

**Titre :** Vers des logiciels antifragiles: perturbations des systèmes logiciels dirigées par la connaissance apprise

**Financement prévu :** Cocher au moins une des cases
- Contrat Doctoral (Ecole Centrale  Lille1  Lille3  Président Lille1)
- Contrat Région avec co-financement: ........................................ (à préciser)
- ANR  CIFRE  DGA  ADEME  Co-tutelle  : ....................................(à préciser)
- Autre: ........................................... (à préciser)

(Co)-Directeur de thèse : Martin Monperrus (HDR 2016)
E-mail : martin.monperrus@univ-lille1.fr
Co-directeurs ou co-encadrants de thèse : Philippe Preux
E-mail : philippe.preux@inria.fr

**Laboratoire :** CRISTAL – UMR 9189
**Groupe Thématique :** Double GL/Dating  **Équipe :** Spirals / Sequel

**Domaine de l'EDSPI :** Automatique (AGITSI)  Informatique

**Descriptif :**
cf pages jointes
Towards Antifragile Software: Knowledge-driven Perturbation of Software Systems with Active Learning

Interdisciplinary topic software engineering - machine learning

M. Monperrus, Ph. Preux

November 30, 2016

Supervision

- Martin Monperrus, Université Lille 1, Laboratoire CRIStAL & INRIA Lille, SPIRALS
- Philippe Preux, Université Lille 3, Laboratoire CRIStAL & INRIA Lille, SequeL

Context

All non-trivial software systems suffer from unanticipated production failures. For instance, on March 12th 2012, the Mozilla Firefox web browser has crashed 270,455 times. Software engineering research invents new concepts to characterize the nature of software failures and devise new frameworks and algorithms to address them. A software failure is commonly defined as an output that does not correspond to the user’s expectations [1]. In the Mozilla Firefox example, the failure is that the program closes (actual behavior) instead of rendering the requested web page (expected behavior). A failure is caused by a fault in the code or an unexpected environmental condition [1]. The classical view on software failures is to combat them with techniques for: detecting faults using static and dynamic software analysis; proving the absence of certain faults with for example theorem provers; model checking the software system; and improving the development processes and tools to prevent the introduction of faults. Those research threads yield fundamental advances in software engineering but fails to eradicate all software failures.

Problem: The fundamental problem is that those approaches are only reactive, consisting in waiting for bugs to happen.

In this PhD, we will study the opposite idea: being proactive with respect to failures [4]. We will devise unconventional techniques that trigger erroneous states and inputs in software systems to study their impact in advance, in a proactive manner.

Let us consider a concrete example. According to our statistics on the Internet, the most common failures in Java software are null dereferences (“null pointer exceptions”) [2]. Null dereferences cause desktop, server and mobile applications to crash on a daily basis. Now, let’s consider a software system that embeds a monitoring system, as well as search-based recovery for null dereferences. As such, the system has already overcome 15 unhandled null dereferences. Now, let us imagine that the system is augmented with a module that selectively injects null values in memory. This system would inject \( x \) (say 3) null values per day so as to 1) assess that the system does not crash upon null dereferences, and 2) pro-actively synthesize and validate null-dereference recovery. On the hardware side of production systems, this idea is already applied. For instance, data centers are regularly subject to power cut so as to assess whether the alternative sources of power are up and running.

Bug detection and bug localization [6] have a very long history in software engineering. An innumerable number of papers deal with these issues; enormous amounts of time and money are spent on these issues. This PhD proposal is driven by the key idea that machine learning and data science more generally are providing renewed insights on these issues. Data science has already brought revolutionary advances in many areas of human activities (science, information management, technology,
commerce, ...). If data science has already been applied to software engineering issues, there remains many problems to study and, we think, many areas in which the current state of the art may greatly benefit from a clever use of data science. From a data scientist perspective, a software is just a data, complex for sure, and evolving along time; from an abstract point a view, a software is an evolving graph, with nodes and vertices being labelled with various sorts of information. Graphs constitute a well-studied type of data in machine learning (e.g. for social networks). However, software graphs are quite different from those usually studied in machine learning, regarding their structure, regarding the information they contain, regarding their evolution, regarding their meaning, and their use.

Scientific Objective

The scientific objective of this PhD is to invent a novel software engineering technique to constantly assess whether software recovery code well handles software failures, in order to obtain more resilient and less fragile software systems.

Methodology

This PhD proposes to apply the scientific method to failure recovery. The scientific method states that all hypotheses must be experimentally validated using falsification experiments. A recovery capability is also an hypothesis: if an event of type \( x \) happens, the system is able to survive. By injecting an event \( x \) and assessing successful failure-recovery, one ensures the truthfulness of the recovery hypothesis in production. In biology, “hormesis” refers to the positive response of biological systems (e.g. a cell) in response to a stressor. This idea of fault injection in production can be seen as the exploration of the notion of hormesis in the domain of software systems.

To devise those systems, the work will be done in two steps. These steps do not have the same duration, the second step taking more time than the first step.

**Step #1:** By injecting failures, there are potential losses due to the propagation of failures to critical components. Examples of loss include loss of money, loss of reputation, loss of lives. In all cases, the losses have to be qualified and a trade-off [7] has to be found between the losses due to the injected failure and the knowledge gained from observing the injected failure in a real setting [7]. For example, knowledge consists of validated recovery, new mined recovery conditions, trigger of better monitoring. The PhD student will create a taxonomy of failures that can be injected in production and the associated potential losses. She/He will start by studying the modern cloud-based software systems: Hadoop, Mesos and Aurora. The applicant will collect a pool of production failures, and systematically analyze them. For each failure, she/he will answer three questions: 1) what is the loss? 2) is it due to the environment, the execution platform, an incorrect piece of code? 3) can the failure be meaningfully injected so as to perturb the system?

**Step #2:** To explore the idea of perturbing production systems with injected failures, the student will set up an “in-vivo perturbation system”, where “in-vivo” means “in production” [3]. This system would be responsible for injecting appropriate failures at the relevant places, according to the benefit and loss characterization resulting from step #1. The system will constantly choose what fault to inject and at which location. Here, we face a problem of sequential decision making under uncertainty: we have to repeatedly decide where to inject a failure, and which kind of failure to inject in order to get a maximal amount of new information about the robustness of the software. Once new information has been gathered, we have to decide where the next failure will be injected, and which, and so on and so forth. This process of repeatedly injecting failures should also take into account the cost of each such injection. This is an exploration/exploitation under a finite amount of resources problem. Bandit algorithms provide a sound way to tackle this sort of problems [5]; they enjoy formal finite-time analysis that provide guidelines for their application, and they are a workhorse for many application, in particular in the field of e-commerce and computational advertising. A contextual bandit would represent a failure to be injected in an abstract space which is the graph of the software. Bandits on graph are an active field of research these days.
Host

The PhD student will work in the CRIStAL laboratory, co-located at the University of Lille and Inria Lille. She/he will be part of both the SequeL team (machine learning) and the Spirals team (software engineering). The applicant will closely interact with the other 15+ PhD students of both teams.

This PhD proposal builds on several years of fruitful collaboration between SequeL and Spirals.

References


