Abstract:

Context & motivation: An automated production system (aPS) is a design-to-order, custom-built system (e.g., a woodworking plant) that is operated over decades and undergoing continuous changes in its mechatronic and software parts [1]. Traceability has long been recognized as key to change management: Identify who needed the change and why, locate where the change should be applied, reason about the change impact, etc. However, the change management support, and for that matter, the traceability itself, has been exclusively scoped within the software engineering artifacts and processes [2]. Question: The challenge facing aPS evolution is to go beyond the software (engineering) boundary because the changes commonly involve both the software (e.g., source code) and the mechatronics (e.g., sensors and actuators). Moreover, the changes of the software and those of the mechatronics are constantly out of sync. For example, replacing a malfunctioned sensor may not trigger any software change, but replacing it with a better one may lead to software updates so that the new sensor’s data could be read and processed more accurately. A root cause of the challenge is that aPS evolution requires different specialities to be communicated and coordinated. At least three types of expertise are needed: mechanical engineering, electrical engineering, and software engineering. Traceability in aPS evolution, then, is truly a systems phenomenon and no longer scoped only within software development. While creating and maintaining traceability is
important, too much traceability can hurt the autonomy of the aPS stakeholders (mechanical engineers, electrical engineers, and software engineers). Thus, not only do we need a new form of systems traceability, but we shall also address such questions as when, who, to whom, and how much associated with systems traceability in the context of aPS evolution. Principal ideas: We envision the systems traceability serves as both an interface and a protocol between software and mechatronic parts of aPS. Our idea is to base Protos [3] to create a representation of aPS traceability. Protos models the requirements of a socio-technical system as commitments between interacting parties rather than the goals of each individual. A socio-technical system involves interactions between autonomous principals facilitated by technical artifacts, including software. Emphasizing interactions, rather than machine/software specifications, has shown to promote openness and accountability in the era of the IoT [4]. Although Protos can handle a single set of changing requirements, its support for continuous aPS changes is insufficient. We therefore propose to augment Protos with specific traceability links with the objective to enhance aPS sustainability. In our vision, the traceability links will not only substantiate the commitment and its refinement, but also enable the identification of conflicts undermining the new needs fulfillment, the reasoning of requirements-induced technical debt, and the creation of innovative requirements for the aPS. Our ongoing work with a bench-scale aPS, namely the PPU (Pick & Place Unit) [5], shows some preliminary results and insights. Contribution: We contribute a formal yet purposeful framework based on Protos for defining systems traceability and using it to sustain aPS changes. The framework contains the formal notations to establish commitments between aPS social entities, the refinements of commitments towards fulfilling the changes, and the traceability as argumentation to guide how the refinements should be carried out. Future directions: Building on and extending our work, we envision the next ten years will shift traceability from inside the software's boundary to the mechatronics that software invariably interacts with and that co-evolves with software.