Science of Computer Programming 97 (2015) 105–112

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Towards evidence-based recommendations to guide the evolution of component-based product families

Leon Moonen

Simula Research Laboratory, Oslo, Norway

HIGHLIGHTS

- The sharing of assets in component-based product families complicates evolution.
- Precise change impact analysis across the complete product family is needed.
- The heterogeneity of artifacts in these families hinders fine-grained analysis.
- We devise recommendation technology that supports accountable software evolution.
- We discuss the research challenges that need to be overcome to build such tools.

ARTICLE INFO

Article history:
Received 25 September 2013
Accepted 5 November 2013
Available online 12 November 2013

Keywords:
Program comprehension
Reverse engineering
Change impact analysis

ABSTRACT

Many large-scale software-intensive systems are produced as instances of component-based product families, a well-known tactic to develop a portfolio of software products based on a collection of shared assets. However, sharing components between software products introduces dependencies that complicate maintenance and evolution: changes made in a component to address an issue in one product may have undesirable effects on other products in which the same component is used. Therefore, developers not only need to understand how a proposed change will impact the component and product at hand; they also need to understand how it affects the whole product family, including systems that are already deployed. Given that these systems contain thousands of components, it is no surprise that it is hard to reason about the impact of a change on a single product, let alone assess the effects of more complex evolution scenarios on a complete product family. Conventional impact analysis techniques do not suffice for large-scale software-intensive systems and highly populated product families, and software engineers need better support to conduct these tasks. Finally, for an accountable comparison of alternative evolution scenarios, a measure is needed to quantify the scale of impact for each strategy. This is especially important in our context of safety-critical systems since these need to undergo (costly) re-certification after a change. Cost-effective recommendations should prioritize evolution scenarios that minimize impact scale, and thereby minimize re-certification efforts.

This paper explores how reverse engineering and program comprehension techniques can be used to develop novel recommendation technology that uses concrete evidence gathered from software artifacts to support engineers with the evolution of families of complex, safety-critical, software-intensive systems. We give an overview of the state of the art in this area, discuss some of the research directions that have been considered up to now and, identify challenges, and pose a number of research questions to advance the state of the art.

E-mail address: leon.moonen@computer.org.

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.11.009

The search for techniques that transcend language boundaries has been a recurring theme in Paul's career. It entered my world when Paul challenged me to investigate solutions for generic data flow analysis as a master student. This paper further explores this theme, driven by the need to better manage the evolution of multi-language software systems. Understanding the interrelations that may interfere with changing such systems requires cross-language analysis. True to form, we use a unified meta-model to represent all relevant information, and build language-independent analyses on top of this model. The application to change impact analysis even remarkably close to the data flow analysis that started all this for me. Paul, I would like to thank you for the inspiration and challenges that led me on this journey.

1. Introduction

Integrated Control and Safety Systems (ICSSs) are complex, large-scale, software-intensive systems to monitor and control safety-critical devices and processes in domains such as process plants, oil and gas production, and maritime equipment. These cyber-physical systems integrate hardware and software components, and they are typically designed as a network of interconnected elements that interact with their environment via physical sensors and mechanical actuators. Consequently, for deployment in concrete situations such ICSSs need to be adapted and configured to the particular safety logic and installation characteristics, such as number and type of sensors and actuators, and the layout and dependencies of safety areas. At the same time there can be considerable similarities between different ICSSs, ranging from high-level requirements to low-level implementation details, in particular for systems in the same domain. For example, consider two off-shore platforms that are alike but not identical in terms of layout or equipment, a very similar observation can be made for their ICSSs. To leverage commonality while accommodating for variations as efficiently as possible, many ICSSs are developed as component-based product families, a well-known tactic to develop a portfolio of software products based on shared assets [1–3].

However, sharing components between software products introduces dependencies that complicate maintenance and evolution of the system because the changes made to address an issue in one product may have undesirable effects on another product in which the changed component is used [4]. Changes arise not only from product-specific improvements, they also originate from rethinking the product family as a whole (known as domain engineering). Shared components can be updated as a result of both *"family-level"* domain engineering activities, as well as product-specific (*"system-level"*) development and maintenance.

For large-scale systems and highly populated product families, a change typically does not come by itself: to achieve a certain effect, developers need to evaluate alternatives for (a) the actual change(s) to be made, (b) additional modifications to address undesired ripple effects of that change, as well as (c) the way in which changes are applied. We refer to these alternatives as evolution scenarios. To choose between alternative evolution scenarios, developers not only need to understand how a change will impact a component and the product in which it is used; they also need to reason how the change will affect the whole product family, including the systems that are already deployed. Given that ICSSs consist of thousands of components, it will be no surprise that it is hard to reason about the effect of a change on a single product, let alone on a complete family of products. Yet, the engineers have to ensure that the components work seamlessly and flawlessly together, also after the proposed changes were made. Despite impressive technological advances, the existing approaches to developing and maintaining software are stretched to the max due to increasing demands, complexity, and scale. Consequently, software is often built and managed based on decisions for which we have insufficient evidence to confirm their suitability, quality, costs, and inherent risks.

Accountable software evolution decisions can be based only on concrete evidence gathered from the actual source artifacts [5]. Change Impact Analysis (CIA) can play a significant role in this process by estimating the ripple effect of a change [6]. CIA takes a set of modifications (the change set), and computes what parts of the program are affected by that change (the impact set) [7]. Source-based CIA estimates the impact of a change by tracking dependencies between program elements. The ripple effects can be found using reachability analysis on this dependence graph. We found that the CIA methods published in scientific literature (and reviewed in the next section) were insufficient for large-scale software-intensive systems and highly populated product families. Challenges include the fact that components can be implemented in various programming languages, and that component composition, initialization and interconnection are typically specified in separate configuration files, ranging from simple key-value pairs to domain-specific languages. This type of artifact heterogeneity complicates many types of system- and family-wide analysis, including change impact analysis [8,9]. Moreover, the conventional methods are aimed at analyzing single systems, and do not take dependencies in a product family into account. A final concern in the context of ICSSs is that the safety-critical nature of these systems requires that they are re-certified after a change. Therefore, cost-effective recommendations should prioritize evolution scenarios that minimize impact scale, and thereby minimize re-certification efforts.

This paper explores how reverse engineering and program comprehension techniques can be used to develop novel recommendation technology that uses concrete evidence gathered from software artifacts to support engineers with the evolution of families of complex, safety-critical, software-intensive systems. Contributions of this paper include an overview of the state of the art in this area and a discussion some of the research directions that have been explored up to now. Next, we identify challenges, and pose a number of research questions that need to be answered to advance the state of the art.