Considering Academia-Industry Projects Meta-Characteristics in Runtime Verification Design

Christian Colombo and Gordon J Pace

Department of Computer Science, University of Malta, Malta

Abstract. Runtime verification, with its practical applicability and myriad of theoretical challenges it still poses, has the potential to bridge the gap between academic research in the field of formal methods with the software industry. In order to facilitate this, it is useful to extrapolate success patterns from previous projects: Are certain characteristics of an industry-academia project a determining factor in the project's success? How can runtime verification design decisions take into considerations project characteristics to improve the chances of success?
This paper attempts to shed some light on these questions by reflecting on five projects with two partners over the past ten years. A number of lessons emerge, perhaps the most poignant one being the need to think long term in setting mutually beneficial goals from which a strong working relationship can emerge.

1 Introduction

Although various underlying notions from runtime monitoring and verification, albeit in limited form, have long found themselves in standard quality assurance practice in industry, its adoption as a first class element and building block of the system being built is still rare. Much of the use of runtime verification techniques in industry, thus still stems from projects in conjunction with academic partners interested in exploring scalability and industrial-relevance of these techniques. In the literature reporting these projects, the focus is invariably the effectiveness of the techniques used on the system under scrutiny. What is usually not reported (being beyond the scientific scope of such publications), is the process of adoption, the context of the project, the logistic challenges encountered and its longer term impact in terms of adoption of techniques beyond the scope of the original project e.g. *Was it an industry- or academia-led project? Was runtime verification being engineered retrospectively on a legacy system or one being developed from scratch?* It is worth noting that some, although not all, of the observations we make are relevant to any project with partners from both industry and academia, and not limited to runtime verification.

In this paper, we present anecdote-based observations based on our experience from five academia-industry projects[1], discussing and reviewing major

[1] While it would have been preferable to include a wider set of projects in our analysis (including those from other research groups), we found that reporting of the "post-

design decisions in runtime verification engineering. An implicit assumption throughout the paper will be that the goal of the academia-industry collaboration is the benefit to both parties. Hence, in the next section we attempt to describe success from the two points of view. In Section 3 we describe project meta-characteristics which might affect the engineering decisions discussed in Section 4. We bring everything together in Section 5 by reviewing our decisions in five projects and report on their success. The last section concludes with some final remarks.

2 Defining Success

Success in the context of industry-academia projects may take various forms, and defies a simple uni-dimensional metric. Instead, it makes more sense to talk of ways in which a project may have a positive impact on the industrial partner, the academic one, or the collaboration between the two.

2.1 Impact on Industrial Partners

We start by identifying different ways in which a collaborative runtime verification project may leave an impact on the industrial partners.

Direct changes within their systems: The direct way in which a project may leave impact is in resulting to changes in their actual software systems (post-project) either through (i) stronger runtime checks within the normal logic of the system to ensure higher confidence in system correctness; or (ii) changes in the system architecture, introducing separate logical units to perform runtime checks or even through the adoption of the use of a runtime verification tool thus completely separating concerns of the normal system and verification logic.

Changes to the quality assurance process: Another way in which projects may leave an impact on the system is through the quality assurance process, particularly during testing, either (i) by enabling the quality assurance team to identify further correctness elements or runtime scenarios, thus resulting in more or better test oracles and test cases; or (ii) through the adoption of techniques from runtime verification to support the specification of a test suite — from the oracles to test scripts and test cases e.g. moving from timeless assertion-based oracles which assert constraints on the system state

mortem" of such projects is sparse in the literature. Many of the observations we make in this paper are on the non-scientific aspects of the research projects (e.g. whether participation of the industrial partner in the project had an impact on the way they approached validation and verification of other systems they were developing), which are typically not discussed in scientific reports of the outcome of such projects. Therefore, while we are aware of several projects which have applied academic techniques in an industrial setting, we cannot include these in this paper due to the lack of information of what happened *after* the end of the project.

at a particular point in the execution of the system, to temporal oracles each of which may refer to and compare the state of the system at various temporal points.

Indirect effects: Although the project may not result in direct changes to the current or planned future versions of the system, or to the quality assurance process, exposure to runtime verification may have indirect and longer-term effects, sometimes due to the Hawthorne effect[2] resulting from oversight by the academic partner due to the collaboration and accentuated by the domain ignorance of the oversight [12]. Two ways we have seen this happen was through exposing architects and developers (i) to alternative ways in which verification can be integrated into a system, yet keeping it in a separate component; and (ii) to the realisation of the possibility to look at different levels of failure handling at runtime going beyond class invariants and pre- and post-conditions.

Although we define the notion of success in a rather broad manner, we have witnessed projects which have partially failed to realise an impact in any of the three effects listed above, typically due to one of (i) the fact that day-to-day fire-fighting with system problems did not leave enough time and resources to consider further changes which runtime verification may require; (ii) lack of interest in immersion into deploying runtime verification due to the designers or developers involved felt that the change was imposed on them (e.g. through company policy to participate in the project) or due to being involved with a team which felt that dynamic analysis was not within their remit (e.g. developers may feel that it is an aspect which quality assurance should handle, or the quality assurance may feel that the onus should be on the developers since the checks are performed at runtime); or (iii) despite an interest to consider the use of runtime monitoring or verification, fear (rightly or wrongly) of its impact on the complexity of the system and its performance impeded its adoption.

2.2 Impact on Academic Partners

Given the different objectives of the academic partners, the impact sought is similarly different. We identify three aspects in which joint runtime verification projects with industrial partners can prove to be fruitful:

Evaluation of new techniques on real-world systems: Typically, a primary measure of success from an academic perspective in many collaborative projects is that of evaluating new techniques on a real-world system. The degree to which the proposed techniques are successfully integrated into the system to be evaluated is a major measure of project impact from the academic side.

[2] Sometimes referred to as the *observer effect*, the Hawthorne effect is the phenomenon that when aware of being observed, individuals may modify aspects of their behaviour.

Application of tools to real-world systems: Many academically developed tools tend to be proof-of-concept artefacts, and mostly developed in an evolutionary manner across generations of students, mostly with little regard to software engineering practice. The experience of applying such tools on real-world systems can be a major challenge but can result in important insight on the strengths but also on design and algorithmic bottlenecks of the tools.

Understanding better the challenges to real-world adoption: The very experience of attempting to transpose runtime verification techniques to be applicable in a real-world setting is in itself a learning experience, exposing the academic partners to real-world challenges, which can lead to the development of new techniques and solutions, and establish longer-term collaboration with the industrial partners.

3 Success-Determining Factors

When considering our past projects (see Section 5), two common success-determining characteristics clearly emerge.

Project lead: One of the determining factors is the degree to which all the partners have at stake in the project. Particularly from the industrial side, where ongoing commercial deadlines and pressures may result in the project being put on the back burner; how central the project is to those partners immediate (or near-future) commercial objectives makes a substantial difference to the chances of success. In general, projects which are instigated, designed and/or led by these partners result in two important advantages towards achieving success:

(i) **Priority:** In industrial settings, priorities may change easily and quickly (e.g., change in leadership, change in market, etc). When a project is low on the industrial partner's priority list, chances are that resources get allocated elsewhere. Projects which were initiated by the industry partner(s) tended to be given more priority and it was easier to obtain information and get access to resources in a timely manner.

(ii) **Engagement:** When an industrial partner gets involved in a project without a clear direct benefit (but perhaps to start a long term research collaboration, to test what academic tools may offer, or to get in touch with students as potential employees), there is a great possibility that the academic researchers will not find much enthusiasm and engagement from the stakeholders within the company — particularly employees who do not see the value of the project.

Legacy vs. new system: A major distinction in industrial runtime verification projects is whether the effort concerns an existing legacy system or whether the system is being designed from the ground up with runtime verification in mind. This issue is particularly pronounced in runtime verification due to the desirability of extracting events non-intrusively *at runtime.*

Legacy systems are typically hard to modify and interoperate with for several reasons, particularly their brittleness and sometimes poorly supported dated technologies. Furthermore, obtaining the necessary information and understanding how a legacy system works might also be significantly challenging due to unmaintained or incomplete documentation and code which has been changed and fixed over and over again.
On the other hand, a system which is being newly built and which incorporates runtime verification from the start, may provide a dedicated interface for the monitor which makes all relevant events readily available and also listens out for any incoming instructions from the monitor.

4 Design Decisions

When applying runtime verification to real-world systems, a number of design decisions have to be taken. In this section we focus on some of the most pertinent ones, particularly those which (from our experience — see the next section) may severely affect project success. It is worth noting that some of the design choices are interconnected and may influence each other.

4.1 System Feedback: Level of Runtime Intrusion

One important decision is that of how intrusive on the system the runtime analysis will be. From a most basic level in which one can merely *monitor* or *observe* a system and log information about its runtime behaviour, then moving up to *runtime verification*, in which not only is the behaviour observed, but particular behavioural patterns are identified to be undesirable and algorithmically classified to be so. This latter level of intrusion can be taken further by adding on logic to support *runtime recovery* or *reparation*, triggering in the case of undesirable behaviour being observed (to make up for it)[3]. One can also go another step further, using *runtime enforcement* [9] to ensure that the undesirable behaviour is avoided in the first place, modifying the system's behaviour to ensure it works as expected.

The higher the level of intrusion, the more difficult it is to have the runtime system to be integrated with the production-ready system. In the context of the success-determining criteria discussed in Section 3, intrusion beyond non-reparatory runtime verification is unlikely to be achieved on legacy systems, but this can be pushed up considerably in industry-led projects on systems still being designed and developed.

4.2 Online vs. Offline

Another fundamental design decision is whether runtime monitoring is carried out online or offline. Online monitoring interacts directly with the system at runtime while offline monitoring involves processing runtime events independently

[3] This is supported by typical RV tools such as JavaMOP [4] and Larva [8].

of the running system. This is a major difference from an intrusiveness point of view as in the case of online monitoring the verification software very likely interferes with the running system and typically competes for the same resources.

Considering the success-determining factors identified in the previous section, similar to higher intrusiveness, online monitoring decreases the likelihood of the project being taken onboard by the industrial partner on the live systems. Therefore, caution should be used and in the case of an academia-led legacy project, it should ideally be avoided altogether. On the other hand, when working in the context of an industry-led project where a system is being designed and developed with online runtime verification in mind, the associated risks can be minimised and catered for.

4.3 System-Monitor Communication

Once the runtime verification mode of online vs. offline is decided, one would typically decide on the communication mode; particularly how the system events are to reach the verifier. The choice is highly dependent on whether monitoring takes place online or offline. As one would expect, offline monitoring allows for communication to be significantly more loosely coupled. For example this may take the form of simply dumping a relevant part of the monitored system's database. The advantage of loose coupling is that it does not interfere with the monitored system. In the case of online monitoring, the choice between tightly and loosely coupled communication modes would typically be more constrained by the desire to have the monitor receive system events in a timely fashion. Options in this case may range from direct method calls from the system, to message transmission over a network. While all these options are possible when considerations are included in the design, more care should be taken when dealing with legacy systems due to repercussions the modifications may bring about.

4.4 Event Extraction

Legacy systems are less amenable to incorporating a clean and modular way of extracting relevant system events. It is for this reason that runtime verification is perhaps one of the best case studies for aspect-oriented programming (AOP)[10]. However, even with the sophistication of AOP, understanding which method invocations to capture and whether these provide enough context to bind the necessary data variables is a non-trivial task. Moreover, when runtime verification is also used to steer the system (as opposed to simply being a passive observer), particular care needs to be taken to ensure that bugs are not accidentally introduced. Maintenance and system updates might also result in unintended consequences in the system-monitor interaction. Another approach to extract events in a legacy system context is through the use of a proxy. This is convenient when the events of interest are visible from a communication point of view.

In contrast, when dealing with a new system, the monitoring of events can be designed as part of the system, i.e. the system proactively makes relevant events available to the monitor. When a new system is designed for an offline setting, one may still opt for a less direct way of extracting events, such as by interfacing with the database. Extracting events from the database would also probably be the most rational choice when dealing with a legacy system, albeit some database modifications might be needed.

4.5 Specification of Properties

When dynamic analysis is used to verify behaviour, the manner in which the discriminator between correct or expected behaviour from bad behaviour is written, plays an important role in determining the success of adoption of the techniques in the real-life system. Although in some projects (particularly academia-led ones) the specification language used may be determined by the objectives of the project itself (e.g. a project focussing on how overheads induced due to monitoring specifications using a particular logic or class of logics can be reduced), in many cases this choice may be flexible, ranging from one extreme of developer-friendly specifications written as observers in the same programming language as the main system, to the other extreme involving the use of complex logics which developers may require training to use in an effective manner. In between, one can find intermediate specification languages which bridge this gap e.g. the use of graph-based formalisms (e.g. automata) or regular expressions with which most developers would be familiar.

As in the other design challenges, the higher the industrial involvement, the more one can choose to identify and adopt an appropriate logic (possibly hidden beneath syntactic sugar or within a controlled natural language), while with lower industrial involvement, developer-friendly formalisms would be preferable.

5 Observations and Commentary

In order to review our past projects in the light of the above design choices, we start by describing the projects. We had five projects with two partners, with the information summarised in Figure 1 ordered in completion date order. The first project was with one partner while the following four projects were with the second partner[4].

1. *System Feedback: Level of Runtime Intrusion.* In the majority of the projects, we opted for the least intrusive of the approaches — that of a passive observer. In the case of the first project where we had online monitoring, we could also alert the system of a violation. Further along the intrusiveness spectrum, the last project includes runtime enforcement where transactions may be stopped if they would lead to a violation. In this last project, the monitor also plays the role of an observer when collating statistics which do not involve corrective actions.

[4] Names of the industrial partners are left out due to information sensitivity and in order to allow us to be able to discuss project success or otherwise more freely.

2. *Online vs. Offline.* The online vs. offline choice is closely related to the intrusiveness choice before. All observer monitors were naturally offline while the rest of the options necessitated an online architecture.
3. *System-Monitor Communication.* The offline monitoring projects made use of a database dump and text files to store the observed data. The online counterparts intercepted method calls in the first case, while in the second case, used asynchronous messages to establish handshakes between the system and the monitor.
4. *Event Extraction.* The second and third projects used a database script to obtain a copy of the relevant part of the database. In the first and fourth projects, we used aspect-oriented programming since these were legacy systems, while in the last project we could construct custom events which were designed as part of the system.
5. *Specification of Properties.* In most projects we used an automata-based formalism — namely DATEs [7]. In the case of the fourth project we used assertions since these were extracted automatically from test traces. Finally, in the last project we provided a controlled natural language [11] which internally compiled to a ruled-based language.

Proj	Part	Characteristics		Decisions				
		Lead	System	Intr	On/Off	Comm	Events	Spec
1	A	Aca	Leg	RV	On	Method call	AOP	Aut
2	B	Aca	Leg	Obs	Off	Db dump	Db script	Aut
3		Ind	Leg	Obs	Off	Db dump	Db script	Aut
4		Aca	Leg	Obs	Off	Text files	AOP	Ass
5		Ind	New	Enf/Obs	On/Off	Async messaging	Custom events	CNL

Fig. 1. Summary of the collaborative runtime verification projects discussed.

Given the above design choices, we can now comment on the successes of each partnership:

Partner A (Project 1). From an academic perspective, this project provided us with the experience and case study needed to create a practical runtime verification tool. From an industry perspective, it was useful to demonstrate to the partner how checks can be embedded in a system and how they can be expressed. However, to date we are not aware of any use of explicit runtime verification technologies adopted within their system.

Partner B (Projects 2–5). The first project with this partner served mainly to prepare the way for other future ones — the next project was an initiative of a number of employees and was successfully deployed on their live data. These experiences led the way to two other projects: Projects 4 and 5. Project 4 was successful from an academic perspective in exploring new ideas however, so far it did not result in direct effects in the industrial technology used. The last project enabled us the freedom of applying the latest academically developed ideas to an industrial system with relative success from both perspectives.

Key Observations

Better chances of success when legacy systems are monitored offline: We note that with the exception of the last project, all projects took place on legacy systems. In all cases with the exception of the first project, we applied offline monitoring when working with legacy systems. The first project which went against this pattern was not successful from an industrial perspective.

Industry-led projects more likely to succeed (industrially): Another observation is that the two industry-led projects were both very successful from an industrial perspective. From an academic perspective, it is less predictable though as the emphasis is on finding a solution to the problem at hand.

Industry-led projects on non-legacy systems give rise to win-win situations: Although the latest project has the advantage of hindsight gained from previous projects, and a well established relationship with the partner, it was also the only one which was both industry-led and dealt with a non-legacy system. We hypothesise that these latter characteristics significantly improved the chances of achieving high levels of success from the point of view of both parties.

6 Conclusions

In this paper, we have considered how poignant characteristics of a project affect the engineering choices in the context of runtime verification. Engineering design decisions are particularly delicate in the context of runtime verification since, unlike other techniques such as testing or static analysis, the generated runtime verification code typically interacts directly with the running system. Runtime verification design aspects such as whether to monitor a system online or offline have been the subject of numerous publications [5, 3, 2]. However, to the best of our knowledge, it is the first time that runtime verification engineering design choices were put against the backdrop of project meta-characteristics. Indisputably, there are various other variables contributing to the success (or lack thereof) of a project. In particular, the human aspects come to mind with questions such as how to build a working relationship amongst partners, how to involve the right stakeholders in the project, and how to handover the outcome of a project to the industrial partner have been tackled in [6] and more generally in [1].

Ultimately, the chance of success of an academia-industry project depends on the quality of the relationship (and the trust) between the parties. Once the parties learn to better understand each other, the chances of success increase dramatically. For this reason, a project in the early years of a collaboration might simply serve the purpose of establishing a good working relationship between the partners. Once the relationship improves, it becomes more probable that undertaken projects are of higher importance to the industrial partner. Hence, the more likely it is that a project is industry-led and concerns a non-legacy system which consequently (as we have explained throughout the paper) increase the design options dramatically.

References

1. Mehmet Aksit, B. Tekinerdogan, Hasan Szer, Hakan Faruk Safi, and Meryem Ayas. *The DESARC method: An effective approach for university-industry cooperation*, pages 51–53. Institute of Research Engineers and Doctors, 1 2015. 10.15224/978-1-63248-038-5-10.
2. Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, pages 17–22, 2010.
3. Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. In *Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM*, pages 15–28, 2017.
4. Feng Chen and Grigore Rosu. Java-mop: A monitoring oriented programming environment for java. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 546–550, 2005.
5. Christian Colombo, Gordon Pace, and Patrick Abela. Safer asynchronous runtime monitoring using compensations. *Formal Methods in System Design*, 41(3):269–294, 2012.
6. Christian Colombo and Gordon J. Pace. Industrial experiences with runtime verification of financial transaction systems: Lessons learnt and standing challenges. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, pages 211–232. 2018.
7. Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *Formal Methods for Industrial Critical Systems (FMICS)*, volume 5596 of *Lecture Notes in Computer Science*, pages 135–149, 2008.
8. Christian Colombo, Gordon J. Pace, and Gerardo Schneider. LARVA — safer monitoring of real-time java programs (tool paper). In *Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009*, pages 33–37, 2009.
9. Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.
10. Gregor Kiczales. Aspect-oriented programming. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, page 730, 2005.
11. Tobias Kuhn. A survey and classification of controlled natural languages. *CoRR*, abs/1507.01701, 2015.

12. Ali Niknafs and Daniel M. Berry. An industrial case study of the impact of domain ignorance on the effectiveness of requirements idea generation during requirements elicitation. In *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013*, pages 279–283, 2013.