Abstract—Software processes evolve as software too. The evolving software processes are adapted to accommodate the rapid progress and changes in software engineering practice. Software process simulation has to evolve as well in order to effectively investigate the evolving software processes. This article illustrates the evolution of software process and the resulting challenges to process simulation with two examples from experiences. From these come a list of recommendations for improving the state-of-the-practice of software process simulation.

Keywords—software process; process modeling; process simulation

I. EVOLVING SOFTWARE PROCESSES

Osterweil shared with us the view that ‘software processes are software too’ [1]. From this perspective, software processes can be regarded as a subset of software, and the evolving software processes conform to the characteristics of an evolving software system identified by Lehman’s Law [2]. They may

- have evolved from legacy process and process models;
- result from a combination of existing process components, process patterns, and/or process fragments;
- be the result of the extension of an existing software process to address new/refined objectives or changes;
- evolve as the result of a need to improve process performance, and ultimately product quality, e.g., CMMI;
- evolve as the result of an intentional change to exploit new technologies, e.g., global software process;
- adapt and evolve on the fly in order to react to changes in the environment or to meet new requirements or constraints, e.g., agile methods.

Since the pioneering work in 1980s [3], particularly the PROSIM\textsuperscript{1} series workshops in the late of 1990s, we have gained a large body of knowledge and experiences that are addressed in hundreds studies dedicated to software process simulation [4], [5]. Nevertheless, many of the reported simulation studies focus certain aspects of conventional software processes. The emerging and evolving software processes, such as global development process, agile methods, and the process changes led by new technologies, are still scarce in PSP community in spite of their popularity in practice.

In the past decade, some existing challenges to process simulation, e.g. model calibration, validation, rapid development and reuse, turn to be more critical and serious in modeling evolving software processes because of the rapidly increasing diversity of software technologies involved in development and the correspondingly growing complexity of software processes. The existing (conventional) process simulation models and modeling approaches do not work effectively.

II. EXAMPLES AND EXPERIENCES

Two examples of simulation modeling of evolving software (and systems) processes are elaborated below in terms of our experiences. They shows the interactions between technology and process in an evolving fashion.

A. System Virtual Integration

Large scale systems development is often a complex undertaking and fundamentally different from that of pure software or hardware systems. Such systems, e.g., modern automotive systems, have to rely on high quality hardware as well as high quality software. In a modern car, almost all functions are electronically controlled and also interlinked. The increasing use of E/E (Electrical and Electronic) systems, particularly software embedded in these systems, determines over 40% of a vehicle development cost.

So far, however, the higher complexity has brought new challenges to automotive development processes that are not well adapted to the needs of software intensive systems of systems. For example, the traditional V-model development process often incurs very high cost during the late verification stage in an ever more global development setting and linger response to the changes from customer and market. A number of new technologies have been invented to tackle these challenges, which eventually change the development process.

Model-based development has been widely employed in automotive development that addresses the heterogeneity of modern vehicle systems and enables a hierarchical design process from an abstract level through iterative refinement by progressively introducing more details. As most of the behavior models are platform independent, however, the entire system (or subsystem) cannot be verified until the integration of these models with the specific platform.

\textsuperscript{1}International Workshop on Software Process Simulation Modeling
System Virtual Integration appears recently as an advanced vehicle development technology extending the conventional model-based development in various aspects. For example, System Architecture Virtual Integration (SAVI) [6] is a verification technique promoted by the Aerospace Vehicle Systems Institute (AVSI). The models, which might be built upon diverse platforms, in their specific languages, and on different refinement levels, are able to communicate with each other within the framework. As a result, it enables model-based virtual integration of vehicle systems (or sub-systems) across the platforms, and further makes the early integrated verification possible.

In our early research project we investigated, modeled, and simulated the possible changes to traditional V-model process enabled by this technology innovation. Fig. 1 shows the corresponding variant of V-model, in which the late integration test is moved early from the right leg to the left.

![Variant of traditional V-model process](image)

In this case, the evolving software and systems technologies enable the evolution of the development process. Although several simulation studies related to the traditional V-model process were reported, they cannot be quickly reused to model this new process variant. The rebuilt model structure is necessary to correctly reflect the changed behavior of the process variant. Also importantly, the empirical data required for model calibration has to be recollected due to the technical advance.

B. Formal Verification Process

The evolution or change of conventional software and systems process also leverages the impact of development technologies.

The high correctness and assurance of software turns to be crucial in the safety- and security-critical systems nowadays. While more commonly applied verification methods (e.g., testing, code inspection and analysis) provide high return for lower assurance levels, they do not scale well to offer high assurance and become prohibitively expensive for that level. Formal software verification, using formal methods, is the verification method that provides the strongest known assurance that a software system implementation is consistent with its specification. However, formal verification is high-effort verification method and the success of applying formal verification on industrial scale systems is rare.

As an exemplar, the L4.verified project successfully completed a large-scale machine-checked formal verification at the code level of the functional correctness of the seL4 (secure embedded L4) operating system microkernel. The project created and applied a middle-out process [7], which is significantly different from conventional software development processes. The success of L4.verified project demonstrates that the evolving software process, without reinventing fundamentally new technologies, is critical in enabling the large scale application of formal verification.

Even though the new process can leverage the application of formal verification, the resulting expense is still significant. The L4.verified project consumed an overall effort of 25 person years. We analyzed the formal verification process and how it influences the rest of the development process based on the precious experiences and data gained from the project. A process simulation study [8] was conducted in order to reflect the dynamic behavior of the process and provide predictive power in support of the planning and execution of future formal verification projects.

In simulation modeling, however, we experienced the challenges and reworks that were seldom addressed in the previous process simulation studies. VPMSim 1.0, the first simulation model of formal verification process, reflects the unique characteristics of formal verification in model structure and simulation results: 1) concurrent development/verification activities, 2) frequent iterations and re-verifications, 3) dynamic and concurrent resource (workforce) allocation, and 4) the effect of invariants in code verification.

The project demonstrates a typical example how the process research and practice work as a technology-enabling factor and benefit the software systems development. Moreover, it shows how the new challenges to process simulation consecutively emerge when modeling such advanced and complicated processes.

III. A Vision of The Future

As the above examples show, software and systems process is continuously evolving in corresponding to the technical advances, varying environments, and business needs. The resulting process is often more complex with diverse variants. As the powerful tool to investigate software processes, software process simulation provides its unique value in investigating and evaluating the evolving software technologies and software processes, but has to evolve as well with the evolving software processes.

The suggested solutions to improving the effective modeling and simulation of evolving software processes are grouped into the following three sectors: repository, methodology, and collaboration.
Repository

A model factory (repository) of process simulators and model components is able to support the model sharing and reuse that benefits both the experienced modelers and newcomers, researchers and practitioners. Even if no similar model can be directly reused in the new process simulation study, a quick inquiry of possible solutions in the archived models is also helpful. Hence more simulation studies on the new and evolving software processes, based on varying simulation paradigms, are encouraged in the community to enrich and broaden our knowledge and experience. The produced new models again can be shared via the model factory, then learned, evaluated and replicated by others.

Lack of data is an a real barrier to the adoption of SPS in practice. Another repository of empirical evidence is needed to support simulation model development and calibration. When the relevant data is missing, the comparable data from similar projects can be used in calibrating model parameters. The simulation outputs can also be validated against the relevant empirical studies.

Methodology

The systematic survey [4], [5] found over a dozen of simulation paradigms ever applied in modeling software process. Although we have some attempt and outcome in maturing the modeling methodology of SPS (e.g., [9]), the community still lacks a series of systematic modeling methodologies formulated to support the rapid development and validation of quality process simulation models.

Given the fact that System Dynamics and Discrete-Event Simulation are relatively popular and mature in SPS [4], [5], we can start with formalizing the methodologies based on our existing experiences, and then set the reference standards for the broader range of simulation paradigms. These methodologies and the associated supporting tools will smooth the steep learning curve of SPS for and novices (especially from industry) and speed up the development of quality simulation models that fit their needs.

In addition, due to the lack of high quality process data in many software organizations, the advance of data mining methods can be applied to configure management systems (e.g., version control systems) for leveraging the adoption of process simulation in practice.

Collaboration

Software processes constantly evolve in real software practice. Process simulation does not provide value if it is separated from practice. The previous success of SPS applications in practice highly relied on the collaboration between academia and industry [10], which becomes more critical than in other areas of software engineering. Researchers take critical roles in enabling SPS technology transfer. A culture and constitution that encourage and facilitate close collaborations between researchers and practitioners have to be established in SPS community.

In support of the collaborations, a mechanism that supports the community building and boosts the communications among the modelers at all levels needs to be formed as well.

Last but not least, simulation modeling of the evolving software processes also relies on the advance in other areas of software process (even software engineering and simulation in other disciplines), for example the metrics for formal verification.

REFERENCES