Actual Process: A Research Program

Lutz Prechelt[1], Sebastian Jekutsch[1], and Philip Johnson[2]

[1] Freie Universität Berlin, Institut f. Informatik, Takustr. 9, 14195 Berlin, Germany
{prechelt,jekutsch}@inf.fu-berlin.de
[2] University of Hawaii, Dept. of Information and Computer Sciences,
POST 307A, 1680 East-West Rd., Honolulu, HI 96822
johnson@hawaii.edu

March 2006

Technical report TR-B-06-02

Abstract. Most process research relies heavily on the use of terms and concepts whose validity depends on a variety of assumptions to be met. As it is difficult to guarantee that they are met, such work continually runs the risk of being invalid. We propose a different and complementary approach to understanding process: Perform all description bottom-up and based on hard data alone. We call the approach *actual process* and the data *actual events*. Actual events can be measured automatically. This paper describes what has been done in this area already and what are the core problems to be solved in the future.

Contents

Freie Universität Berlin

1 Introduction

Most process research[1] is concerned with highly compound phenomena and hence heavily relies on abstraction. This is unavoidable, as the level of detail in these phenomena would be unmanageable without abstraction. However, abstraction unavoidably implies making assumptions. And indeed one finds that current software process research is full of assumptions. Let us illustrate this statement with a fictitious example:

> Assume we are investigating a proposal for processes in which time-to-market is critical. The proposal recommends to release the software from testing as soon as the number of new defects detected per day consistently drops below some threshold. The *explicit* assumption underlying this proposal claims that dropping below the threshold means that (a) most relevant defects have been found, and (b) the remaining ones are sufficiently subtle (and failures sufficiently rare) that users will not be bothered unacceptably. Proposition (b) can be validated empirically, which is exactly what good process research would attempt to do.
>
> However, there are many further *implicit* assumptions that underlie the proposal. For instance: The testers' motivation is constant over time; test effort per day is non-decreasing; defects found but not yet removed do not obstruct the detection of further defects; the criteria for flagging some behavior as a failure are constant over time; defects do not cluster. Any of these implicit assumptions may be wrong from time to time, making it very difficult to validate the proposal, find an appropriate threshold, or reproduce the findings in a different project.

We propose to complement this style of assumption-based process research by another style of process research that attempts to be assumption-free as far as possible. This style is completely bottom-up, data-driven, and thus initially very low-level. It attempts to find and understand what we call *actual process*.

This paper will now first introduce precise terminology for talking about actual process and for contrasting it to assumption-based process (Section 2), and will then describe how this relates to various areas of existing process research (Section 3). The heart of the paper then sketches what we believe should be the agenda for actual process research (Section 4) and the contributions made so far (Section 5).

2 Terminology

A *process model* is a set of concepts that can be used for describing or prescribing software development. Each concept is given by an intensional definition and involves abstraction, i.e. leaves out detail. Since the real world, to which the definition applies, is not constrained to the ideas mentioned in the intensional definition, the extension of the concept is not known; there exists an infinite set of instances with surprising features. Example: For the concept "devising a test case" there may be instances in which a software engineer devotes a great deal of thought to the structure and content of the test case, and other instances dominated by

[1] By process we always mean software engineering process, but the ideas in this article can readily be applied to systems engineering as well.

carelessness. The difference may be relevant for the process but is completely ignored by the definition.

Most process research is concerned primarily with such intensional concepts (and hence process models); it either talks about the concepts as such, or about instances of these concepts. A concrete process execution described by instances of such concepts we call a *conceptual process*.

In contrast, *actual process* is a description of a software development that uses extensional concepts only, i.e. concepts are exclusively formed by grouping instances. Such concepts are just designations and do not involve any abstraction whatsoever.

The basic notion of actual process is the *actual event*, an individual, objective, directly detectable fact ("X happened at time Y") involving no or almost no interpretation. Note that actual events are always instances, not concepts.

Arbitrary sequences (possibly of length 1) of actual events may be considered an instance of some concept, for example an activity. In principle, forming this kind of concept is a purely syntactical operation: assigning a name to a set of such actual event sequences. In practice, however, this is obviously an interpretation step: We assign such a name to such a set of sequences that the resulting concept resembles one of the abstractions we might have in a process model. The difference is that the, say, *actual activity* thus formed still carries all the detail with it that is contained in the individual actual events and hence provides a much richer picture of reality.

Likewise, we mirror all common process terminology: Process models and conceptual processes talk about *conceptual events, conceptual actions, conceptual activities, conceptual document structures, conceptual transition criteria* etc., while actual processes talk about *actual events, actual actions, actual activities, actual document structures, actual transition criteria* etc.

Concepts in conceptual process often involve ideas such as intent or goal that cannot be observed; they hence make assumptions. (They regularly also make assumptions about things that can or could be observed.) In contrast, concepts in actual process are based on actual events alone and are therefore assumption-free. Note, however, that the *use* of actual process concepts *does* involve assumptions, namely regarding usefulness, insights etc.

3 Related Work

The idea of bottom-up, data-driven process understanding is of course not at all new. Several related strands of work exist and this section will explain their relationship to actual process research.

Personal Software Process (PSP). The PSP [6] is a methodology that attempts to transfer the process maturity ideas of the CMM-SW to the level of individual engineers. It contains defined process (as in CMM level 3), process measurement (level 4), and learning from experience for continuous process improvement (level 5). For two of the authors, the PSP was the stimulus that led to actual process: They initially considered the PSP a good idea (at least the measurement and improvement part), but then their research found that the high levels of discipline and data collection effort required to run PSP made its adoption unrealistic [8,15]. The ideas and work around actual process have been based in part on these insights. For example, rather than requiring software engineers to manually maintain logs of effort as in the PSP, the actual process approach suggests that the software engineers use “sensors” attached to development tools to automatically collect data on their development activities.

Software archeology and mining software repositories. One source of actual event data is repositories that are routinely formed during software development, most importantly version management and defect tracking histories. Such data have been used for process-related research many times (e.g. defect prediction [3] or identifying suspect processes [4]), but are suitable only for certain narrow purposes; general process analysis is usually not possible from merely this kind of information [19], as it is too coarse-grained and one-dimensional. In actual process research, such data is augmented with other, finer-grained events in order to produce a more multi-dimensional representation of development.

Software metrics. Actual process and software process metrics share the approach of reflecting the process in data taken directly from process reality. The difference is that in comparison to most actual events, many (though not all) process metrics are relatively high-level, and so tend to contain or reflect assumptions. The intended interpretation of the measurement is valid only if the assumptions are met, but the measurement does not provide sufficient information to check whether this is the case. Therefore, such metrics are elements of assumption-based process models. A typical example of a metrics assumption would be "the measurement comes from a process that was performed as prescribed by the process definition" – which is the reason why the CMM introduces metrics only on level 4, after defined processes have been introduced on level 3.

Work studies of programmers. A number of in-vivo studies have been made on how programmers spend their time during coding [14], what they actually do while maintaining and comprehending code [12], or what impact interruptions [1] have on working time and activities. These studies use live observation, so that the findings are based on actual process. However, data collection is usually based on manual recording and is therefore very expensive and possibly inflicted by selection bias (and also self-perception bias due to the use of “loud thinking” [17]). Actual process research promises to allow for performing similar studies on a massive scale. A similar statement applies to the related laboratory research under the heading of cognitive psychology, such as [2,5].

Usability research and assessment. When usability researchers or professionals assess the usability of a software product, they use an approach fairly similar to actual process research: observe fine-grained, low-level events and analyze the event streams for recurring behavior patterns that indicate usability problems; use these insights to suggest product improvements [13]. Two important differences to actual process research exist. First, the target of the analysis is a software product (rather than a process), whose characteristics are much more concrete and fixed, which makes it an easier target for understanding. Second, the density of "interesting" events in the observation is usually much higher so that manual analysis (and even "manual observation") is a viable option and is in fact a common approach.

Microprocess and macroprocess research. In this line of research, macroprocess refers to the gathering of externally observable outcomes of processes—time taken, costs incurred, and so forth, with the goal of determining how changes in resources might affect these outcomes. Microprocess research attempts to provide process descriptions (written in suitable formal notations) that can be used to explain how and why macroprocess interventions yield the results that they do. In other words, microprocess descriptions provide causal theories for macroprocess behaviors. In due time, research on actual process should become a primary source of input for microprocess research.

4 Research Program

Actual process research will start by collecting actual event data (both manually and automatically) and analyzing it manually in order to identify meaningful descriptive low-level concepts. These concepts serve as the foundation for automating data collection and automating inference of mid-level behavioral descriptions and patterns. These descriptions serve as the foundation for then bridging from actual process research to conceptual process research: validating assumption-based process models, assessing process quality, and validating process improvements.

Here is our idea of the relevant research tasks and questions in a little more detail. Their execution will be iterative in the form of a set of intertwined feedback loops.

1. *Identify a basic vocabulary*. We need a set of low-level, assumption-free descriptive concepts that allow to bridge the gap between the basic, super fine-grain (and hence almost meaningless) actual events and a level of low-level actual process phenomena about which humans can think, formulate expectations and assumptions, and check their validity. See Section 5.3 for some current work in this area. Since some aspects of the development process are much more difficult to observe than others, initial research will focus on concrete technical process areas such as programming, testing, debugging, configuration management, and the like, and will avoid fuzzier areas such as most requirements work, much design work, all kinds of meetings, and others. Qualitative research methods will have to be used when performing this step.
2. *Automate data collection and management*. Full-scale application of actual process is impossible without automated data collection. Suitable tools need to be developed. Their core concept is the *sensor*, a software or hardware/software component for detecting one type of actual event. See Section 5.1 for current work in this area.
3. *Find relevant events*. Many potential kinds of actual events will turn out to be hardly helpful in understanding actual process. Furthermore, sensors for many kinds of events will be very difficult to build. We need to find out which of these are worth building and which of the simpler ones are the most useful. See Sections 5.1 to 5.4 for some current work in this area.
4. *Learn how to cope with imperfect sensors*. Actual events are objective facts. However, a sensor for detecting such events may flag spurious ones, miss real ones, or distort event parameters. What does this mean for further analysis in practice? Can it still be called assumption-free? How do we cope?
5. *Develop robust inference mechanisms*. We need methods for efficient analysis of actual process data in order to find instances of the low-level concepts, of known mid-level event patterns (and thus behavior patterns), and of candidates for new patterns – all this despite the presence of a very high fraction of irrelevant data.

Such methods may range from straightforward pattern matching based on regular expressions to techniques based on reinforcement learning. See Section 5.4 for some current work in this area.

6. *Identify common behavioral patterns (1)*. Replace the current naïve textbook-style descriptions of process activities by realistic, ecologically valid ones that reflect the full complexity and variability of an actual development process.
7. *Identify common behavioral patterns (2)*. Identify recurring behavior that does not map to commonly used process abstractions and form new actual process concepts for characterizing it.
8. *Infer intent*. When and how is it possible to reliably diagnose the goals and intent of an actor given only a stream of actual events? Understanding intent is important for mapping actual process descriptions onto process models and detecting relevant gaps in the latter.
9. *Validate process models*. We need to develop methods to use actual process data for assessing the accuracy of a given process model. How can we systematically decide which deviations are important and which should be ignored?
10. *Assess process quality*. How can we use actual process data for inferring the quality, productivity, and risk attributes of the overall process? See Section 5.2 for some current work in this area.
11. *Validate process improvements*. How can we use actual process data for quantifying the changes of quality, productivity, and risk attributes between the current process and a previous version of it?

Besides the scientific and technical hurdles, there are also social ones. For instance, actual event collection will often raise privacy concerns. Our current approach for overcoming them is to provide each engineer with complete control over the data relating to him or her, which restricts somewhat the kinds of data one may collect.

5 Contributions So Far and Current Work

This section summarizes the contributions already made to actual process research and the work currently underway.

5.1 Event Collection Tools

The foundation of any practical actual process work is the collection of actual event data. Several tools for automatically collecting data from a variety of sources have already been built.

Hackystat [8] is a rather general framework for automated data collection. *Sensor* components detect arbitrary events and transfer them to a server. There are many different sensors. Current ones are embedded into tools, e.g. many sensors such as "user is reading file X" and the like have been integrated into various IDEs, word processors, and web browsers. Future sensors may also be stand-alone, e.g. a "user is present/absent" sensor could be based on the image supplied by a webcam or on the proximity detection of a Bluetooth mobile phone. The server collects data from many sensors, stores it, and provides various kinds of queries, summaries, and displays for analyzing the data. PROM [18] is somewhat similar to Hackystat.

ElectroCodeoGram [16] uses a similar approach (and in fact the very same sensors), but with a different focus, namely the collection of still more fine-grained data and real-time pattern detection in event streams.

5.2 Software Project Telemetry

The telemetry project [9] attempts to find out how to discriminate between the *healthy* and *crisis* states of a development project based on actual process data (collected by Hackystat) only. The goal is automatically detecting early hints when a healthy project is about to go into crisis; the primary method is observing in-project trends. So far, the analysis is not fully automated but rather presents a number of graphical summaries to a human project manager.

Compared to seemingly similar "project dashboard" applications, the telemetry approach builds on a much broader set of data with correspondingly greater potential of uncovering relevant process phenomena at an early stage. Furthermore, the automated collection of the data makes frequent modifications to the graphical summaries viable and supports a learning-from-experience feedback cycle.

5.3 Error Detection and Prevention

The error prevention project [7] attempts to find ways of using actual process data to automatically flag dubious areas in work products as "likely to be wrong". However, in contrast to quality assurance approaches using some kind of automated static or dynamic analysis we do not analyze the work product but rather its construction process.

We are currently trying to operationalize behaviors that are known to be highly error-prone. It is known that mental overload, vague knowledge, guessing, as well as uncomfortable or disruptive working conditions are root causes for increased probability of making errors and we conjecture that they can be diagnosed without any semantic knowledge about what problem is currently being solved, because they will be reflected in actual process patterns such as chaotic code construction, interruptions, frequent redesigns, missing tests, trial-and-error episodes, etc. General kinds of error-prone programming behavior, such as copy-paste episodes while changing code [10], are also studied.

We expect to uncover actual process patterns that have high predictive power for defects in code or other software development documents.

5.4 Diagnosing Specific Expected Behaviors

The Software Development Stream Analysis Project [11] explores the use of rule-based approaches for inferring higher level developer behaviors from low-level actual process event streams. Its initial application area is to capture sequences of developer actions within the Eclipse IDE and infer from these sequences whether or not the developer is using the Test-Driven Design (TDD) development method or not. We will validate the inference process with orthogonal measurements such as video capture. Once validated, such a mechanism can be used to both assess compliance with a prescribed or expected process and provide insights into when and where that process is helpful.

6 Conclusion

We believe that actual process research promises to narrow the gap between process models and actual development behavior. It will initially support the thorough investigation of certain important person-level aspects of software processes, such as interruptions, errors, task costs, action frequencies, and detailed-process compliance. In the longer run, it will provide a much sounder empirical basis for many aspects of current process models by allowing to refine increasingly higher-level concepts and making them more ecologically valid.

The foundations have been laid, both conceptual (as described in this article) and technical (represented by process measurement tools). Now we need to build on them.

Acknowledgements
We are grateful to Lee Osterweil for helpful discussion about the term 'microprocess'. Furthermore, we would like to thank all those involved in the various research projects as programmers or users for their time and effort.

References

1. Burmistrov, I. and Leonova, A: Do interrupted users work faster or slower? The micro-analysis of computerized text editing task. In: J. Jacko and C. Stephanidis (Eds.) Human-Computer Interaction: Theory and Practice (Part I) – Proceedings of HCI International (2003), Vol. 1. Mahwah: Lawrence Erlbaum Associates, 621-625
2. Davis, Simon P.: Models and theories of programming strategy. Int. Journal Man-Machine Studies (1993) 39, 237-267
3. Graves, T. L., Karr, Alan F., Marron, J. S., Siy, H.: Predicting Fault Incidence Using Software Change History, IEEE Transactions on Software Engineering, vol. 26, no. 7 (2000), 653-661
4. Hassan, A.E., Holt, R.C.: The Chaos of Software Development. Proc. International Workshop on Principles of Software Evolution, Helsinki, Finland, (2003), p. 84
5. Hoc, J.-M., et.al (ed.): Psychology of programming. Academic Press 1990
6. Humphrey, Watts S.: Using A Defined and Measured Personal Software Process, IEEE Software, vol. 13, no. 3 , (1996), 77-88
7. Jekutsch, Sebastian: Utilizing the Micro-processes of Software Development for Defect Prevention, Technical Report B-04-15, Inst. f. Informatik, Freie Universität Berlin, 2004, p.29-32 (http://projects.mi.fu-berlin.de/w/pub/Main/SebastianJekutsch/phdworkshop04.pdf)
8. Johnson, Philip M.: Project Hackystat: Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis, Technical Report csdl2-01-13, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, November 2001
9. Johnson, Philip M., Kou, H., Paulding, M., Zhang, Q., Kagawa, A., Yamashita, T.: Improving Software Development Management through Software Project Telemetry. IEEE Software, vol. 22, no. 4 (2005), 76-85
10. Kim, M., Bergman, L., Lau, T., Notkin, D.: An Ethnographic Study of Copy and Paste Programming Practices in OOPL, International Symposium on Empirical Software Engineering, August 2004, pp. 83-92
11. Kou, Hongbing: Results of a Pilot Study to Validate the Zorro System for Test-Driven Development Recognition using Software Development Stream Analysis, Submitted to the 2006 Software Process Workshop, Shanghai, China, May, 2006.

12. von Mayrhauser, A., Vans, A.M.: Identification of Dynamic Comprehension Processes During Large Scale Maintenance. IEEE Transactions on Software Engineering, vol. 22, no. 6 (1996), 424 - 437
13. Nielsen, Jakob: Usability Engineering, Morgan Kaufmann, 1994.
14. Perry, D. E., Staudenmayer, N. A., Votta, L. G.: Understanding and Improving Time Usage in Software Development. At&T BellLaboratories, USA, Massachusetts Institute of Technology Sloan School of Management, USA, 1995.
15. Prechelt, L., Unger, B.: An Experiment Measuring the Effects of Personal Software Process (PSP) Training. IEEE Transactions on Software Engineering 27(5):465-472, May 2001.
16. Schlesinger, Frank: Protokollierung von Programmiertätigkeiten in der Eclipse-Umgebung. Diploma thesis, Inst. f. Informatik, Freie Universität Berlin, 2005
17. Seaman, Carolyn B.: Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering, vol.25, no.4 (1999), pp.557-572
18. Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. Proceedings of the 29th EUROMICRO Conference, Belek-Antalya, Turkey, September 01 - 06, 2003
19. Wolf, A. L., Rosenblum, D.S.: A study in software process data capture and analysis. Proc. Second International Conference on the Software Process, IEEE Computer Society, Feb. 1993, pp. 115-124