Leap: A "Personal Information Environment" for Software Engineers

Philip M. Johnson
Collaborative Software Development Laboratory
Department of Information and Computer Sciences
University of Hawaii
Honolulu, HI 96822 USA
johnson@hawaii.edu

ABSTRACT
The Leap toolkit is designed to provide Lightweight, Empirical, Anti-measurement dysfunction, and Portable approaches to software developer improvement. Using Leap, software engineers gather and analyze personal data concerning time, size, defects, patterns, and checklists. They create and maintain definitions describing their software development procedures, work products, and project attributes, including document types, defect types, severities, phases, and size definitions. Leap also supports asynchronous software review and facilitates integration of this group-based data with individually collected data. The Leap toolkit provides a "reference model" for a personal information environment to support skill acquisition and improvement for software engineers.

Keywords
Software developer improvement, metrics

1 INTRODUCTION

The demands of software development on "Internet Time" include shortened time to market, reduced development budgets, and faster release cycles. The pace of technical and economic innovation in Internet Time industries tends to result in increased organizational volatility, including frequent restructuring and high employee turn-over. These combined pressures can wreak havoc with traditional, top-down process improvement initiatives, which typically require: (a) sustained commitment from top-level management for years at a time; (b) "champions" who remain within the organization with stable responsibilities; and (c) a stable developer, product, and market focus so that any process improvement opportunities identified during one product or development cycle remain relevant during the next. Finally, top-down process improvement initiatives tend to incur significant financial and administrative costs to implement and administer the program, report its findings, and justify its continued existence.

Empirical top-down process improvement initiatives must combat an additional problem: measurement dysfunction. Research by Robert Austin on software development organizations identifies measurement dysfunction as a significant obstacle to process improvement [1]. Measurement dysfunction refers to a situation in which the act of measurement affects the organization in a counter-productive fashion. Such dysfunction occurs because many process measures have two potential applications: (1) to provide *information* to the organization and (2) to support *performance evaluation* of individuals and groups. Since it is impossible for an organization to guarantee that a measure, once collected by the organization, will never be used for performance evaluation, process measures may be skewed to reflect what the organization (or process improvement team) wants or needs to hear, rather than what is actually occurring in the organization.

Despite these concerns, traditional top-down process improvement initiatives remain an important and valuable component of a high quality software development organization. However, it is also possible to pursue a "bottom up", developer-centered approach that addresses many of these concerns. In a bottom-up approach, the focus is on providing individual developers with the insights necessary to acquire and improve their technical skills. Management buy-in and support becomes secondary to the developers' self-interest in their own professional development. Management reports on the progress and success of the individual's skill acquisition efforts are no longer required and in fact counterproductive, since preserving the privacy of personal measurements and insights is crucial to preventing measurement dysfunction. Finally, the tendency of modern software developers to frequently change organizations can undermine their commitment to top-down process improvement initiatives, while a bottom-up approach represents a "portable" activity that the developer can maintain across jobs and organizations.

For two years, we have pursued a research initiative regarding bottom-up technical skill acquisition and improvement called Project Leap. We hope through this research initiative to uncover some of the principles underlying successful bottom-up process improvement. Project Leap leverages our prior research experiences in formal technical review [6] and the Personal Software Process [3]. Based upon these expe-

riences, we conjecture that approaches to bottom-up process improvement are made more effective by obeying the four "Leap" design constraints:

- **L**ightweight. Bottom-up methods should be lightweight. In other words, they must involve a minimum of process constraints, be easy to learn, be easy to integrate with existing methods and tools, and require minimal investment and commitment from management. If a bottom-up method imposes new overhead on a developer, then that effort should yield a short-term, positive return-on-investment to that same developer.
- **E**mpirical. Bottom-up methods should have a quantitative, as well as qualitative dimension. A lightweight orientation cannot be gained at the expense of high quality collection and analysis of data. Developer improvement should be observable over time through measurements of effort, defects, size, and time, in combination with improvements in checklists, patterns, and so forth.
- **A**nti-measurement dysfunction. Measurement, while an integral part of effective bottom-up methods, should be carefully designed to minimize dysfunction. Yet the most simple solution to dysfunction—making all data totally private—is incompatible with the benefits to the organization of sharing certain kinds of data and process improvements. A goal of Project LEAP is to find a suitable balance between public and private measurement data.
- **P**ortable. Useful developer improvement support should not be tied to a particular organization such that the developer must "give up" the data and tools when they leave the organization. Rather, this support should be akin to a developer's address book; a kind of "personal information environment" for their software engineering skill set that they can take with them across projects and companies.

These four criteria, when composed together, create additional requirements. For example, we believe that extensive automation is required for any method that is simultaneously lightweight, empirical, and anti-measurement dysfunction. On the other hand, automation clearly does not guarantee lightweight processes or meaningful empirical evidence of improvement. As an example, one criticism of our CSRS automated review system [5] was that its extensive measurement system would lead to dysfunctional behavior in an industrial setting.

Our efforts in Project Leap have produced a toolkit which has been in public release for approximately one year, and in active classroom and research use for approximately six months. The Leap toolkit is also under small scale evaluation at two of our industrial affiliate sites, and we intend to pursue broader industrial evaluation over the coming year.

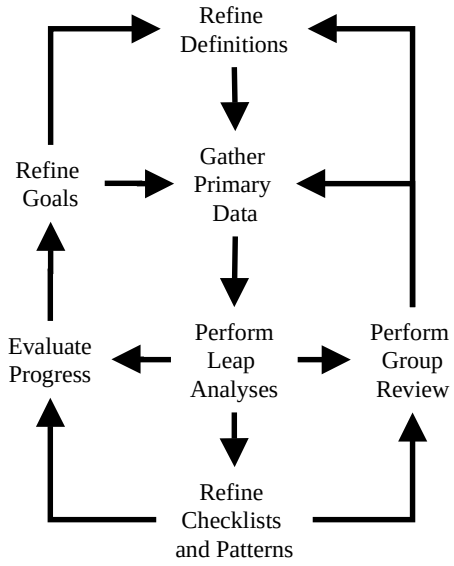

Figure 1: Paths of data collection and analysis in Leap. Several potential entry and exit points exist.

Indeed, our motivation for this research demo presentation is to introduce the toolkit to a broad audience and solicit increased involvement in its evaluation from the research and industry communities.

The Leap toolkit is implemented in 100% Java and runs on most platforms. The most recent release is available for download from our research group home page at http://csdl.ics.hawaii.edu/.

2 The Leap method

The tools in the Leap toolkit are designed to support the following general paths of personal and review-based data collection and analysis as illustrated in Figure 1.

In Leap, there are two "central" activities: gathering primary data and performing Leap analyses. These can be augmented by secondary activities of refining definitions, checklists, and patterns. Finally, these central and/or secondary activities can be directed toward individual skill acquisition and improvement or group review of work products. The following paragraphs provide a bit more detail:

- *Generate or refine goals for technical skill acquisition and improvement.* Example goals could include improved estimation of size or time, improved skill at upstream design, increased direct hours on major tasks, decreased incidence of certain classes of defects, etc.
- *Generate or refine definitions of projects, defect types, document types, etc.* Leap definitions provide two benefits: (1) they reduce the overhead of data collection (by providing pull-down or pop-up menu support for definition usage), and (2) they aid in analysis (by ensuring common terminology between projects).

- *Collect primary data on size, time, and defects.* The basic Leap toolkit has been augmented with specialized tool support for in-process time data collection, in-process defect collection, and with a tool for hierarchical, grammar-based size calculation.
- *Obtain additional defect data via group review.* Leap builds in support for asynchronous review and dissemination of review artifacts via the web and/or email.
- *Perform analyses on primary data.* Leap builds in dozens of analyses accessible through pull-down menus, including charts and tables providing project summary statistics on time, size, and defects; defect types, rates (size/time, defects/time), densities (defects/time, defects/size), trends, and planning/estimation tool support.
- *Evaluate progress toward goals.* Leap analyses provide software engineers with quantitative data that they can use to determine if they are making progress toward their goals. Sample goals include targets for direct hours applied to given projects, reduction in the frequency and/or expense of certain types of design or implementation defects, improvement in review efficiency and/or effectiveness, and improvement in the accuracy of planned size and time values. Figure 2 illustrates one example analysis chart for planning.
- *Generate checklists and patterns.* To support defect prevention and encode "best practice", Leap enables developers to generate checklist items and simple process pattern information.

Leap is similar to other formulations for individual and group process improvement in software engineering, such as Goal-Question-Metric (GQM) [2] and the Personal Software Process (PSP) [4]. It is the attempt to satisfy the Leap constraints in a bottom-up context that produces differences between the Leap toolkit and many other approaches.

First, Leap does not record authorship or other identification; all data collected and manipulated in Leap is unattributed. In a personal environment, authorship is unnecessary, and lack of attribution is a small but important step toward decreased measurement dysfunction.

Second, while Leap provides various definition mechanisms to enable developers to describe what kind of procedures they used to develop a work product or perform a review, Leap makes no attempt to enforce or assess compliance with a particular procedure. Indeed, Leap recognizes that useful definitions must be "bootstrapped" over time through the use of the tool.

Third, Leap enables developers complete control over what kinds of Leap information is shared with others. While developers typically are happy to share certain kinds of insights, such as checklists and patterns, we have found that time data is especially susceptible to measurement dysfunction. Thus, for example, Leap makes it easy for developers to keep track of time they devote to a review activity, but provide to others only the set of defects they uncovered during review.

Fourth, Leap provides an integration mechanism for both personal software engineering data and data generated through the process of group review.

Leap is similar in many ways to the Personal Software Process (PSP). The essential differences between Leap and the PSP are as follows. First, the PSP views automated support as helpful but optional. In contrast, Leap views automation as essential to reducing the overhead of process data collection and analysis to an acceptable level. Our prior research also indicates that automation may be necessary (though not sufficient) for collection of accurate personal process data [3]. Second, the PSP involves an essentially "heavyweight" orientation toward process definition and adherence: in the PSP, one is instructed to follow quite rigidly defined process scripts which sometimes involve practices quite unfamiliar to most developers (such as to completely code all system definitions before compiling for the first time). Leap allows a more "lightweight" orientation, in which one can begin collecting and analyzing data without a great deal of process definition, adding such definitions incrementally when deemed useful. Third, unlike PSP, Leap integrates support for asynchronous review as an essential service in the toolkit. Fourth, the PSP requires you to collect data on your defects—what you do wrong. In addition to defects, Leap also helps you to collect data on your patterns—what you do right.

STATUS

Leap has been in internal and public release for approximately one year, and we are now seeking greater external use in academic and industrial settings. We provide three paths for extension of the platform. First, the Leap data file specification is a simple, restricted HTML format, which facilitates interoperation of Leap with other software engineering tools at the file level. Second, Leap includes an "extensions" mechanism, which allows developers to write Java code that can be dynamically linked to the Leap at invocation time and allow third-party Java tools to extend Leap with new menus and applications. Finally, we intend to make an open source version of the Leap tool kit available for direct modification, experimentation, and enhancement by the software engineering community.

ACKNOWLEDGMENTS

I gratefully acknowledge my colleagues in the Collaborative Software Development Laboratory (Cam Moore, Robert Brewer, Jennifer Geis, Joe Dane, Jay Corbett, Anne Disney, and Russ Tokuyama). This research was sponsored in part by grants CCR-9403475 and CCR-9804010 from the National Science Foundation.

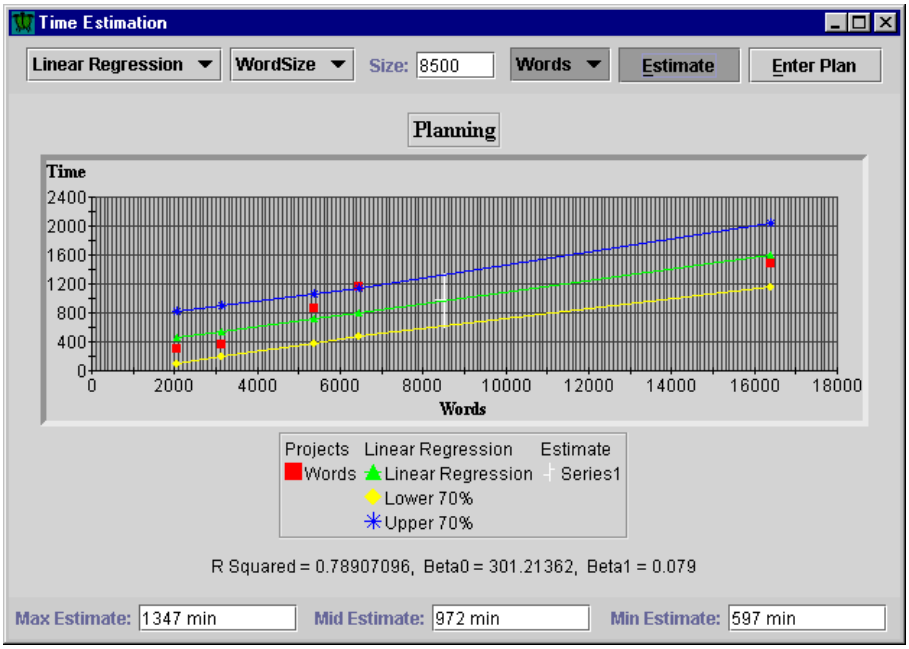

Figure 2: A sample analysis in Leap allowing the developer to use historical data to estimate the time required for a new project. The tool produces estimates by applying either linear regression or average/min/max analyses to historical data. The size metric used is user-definable.

REFERENCES

[1] Robert D. Austin. *Measuring and Managing Performance in Organizations*. Dorset House Publishing, 1996.

[2] Victor Basili and David Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6), November 1984.

[3] Anne M. Disney. Data quality problems in the Personal Software Process. M.S. thesis, University of Hawaii, August 1998.

[4] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, New York, 1995.

[5] Philip M. Johnson. An instrumented approach to improving software quality through formal technical review. In *Proceedings of the 16th International Conference on Software Engineering*, pages 113–122, Sorrento, Italy, May 1994.

[6] Philip M. Johnson and Danu Tjahjono. Does every inspection really need a meeting? *Journal of Empirical Software Engineering*, 4(1), January 1998.