"How Do I Evaluate THAT?" Experiences from a Systems-Level Evaluation Effort

Pardha S. Pyla, H. Rex Hartson, Manuel A. Pérez-Quiñones, James D. Arthur, Tonya L. Smith-Jackson, and Deborah Hix

Center for Human-Computer Interaction, Virginia Tech
Blacksburg, VA 24060 USA
{ppyla,hartson,perez,arthur,smithjack,hix}@vt.edu

Abstract. In this paper we describe our experience deriving evaluation metrics for a systems-level framework called Ripple that connects software engineering and usability engineering life cycles. This evaluation was conducted with eight teams of graduate students (falling under four types of development models) competing in a joint software engineering and usability engineering course to create a software solution for a real world client. We describe the challenges of evaluating systems-level frameworks and the approach we used to derive metrics given our evaluation context. We conclude with the outcome of this evaluation and the effectiveness of the metrics we employed.

Keywords: Systems-level evaluation, evaluation metrics, goal-question metric.

1 Introduction and Background

1.1 Types of Evaluation

One of the fundamental activities in interaction development is evaluation. Evaluation is multi-dimensional, and along one dimension within human-computer interaction there are two kinds of evaluation: formative and summative. Formative evaluation, conducted during a development effort, is used to support human-computer interaction (HCI) *practice* (identifying design flaws in the existing state of the design). On the other hand, summative evaluations, predominantly conducted after the design is implemented, are used to support HCI *research*.

Typically, summative studies are about comparing different systems by controlling certain variables and observing their effects to make conclusions about which is "better" (on some measurable scale) within a narrowly defined criterion. These kinds of studies are one of the few staple formal methods in an otherwise mostly ad-hoc research domain of HCI. They provide formulaic answers to particular research questions where measurable independent variables can be isolated and measured. Furthermore, because these studies are relatively straightforward to conduct and work well for such targeted contexts, they are popular in HCI literature. However, these studies have proven difficult and impractical for answering the larger, less-constrained research questions in systems-level research [1, 2].

M. Kurosu (Ed.): Human Centered Design, HCII 2009, LNCS 5619, pp. 292–301, 2009.

1.2 Introduction to Ripple: A Systems-Level Research Effort

Interactive software systems have both functional and user interface (UI) components. UI design requires specialized usability engineering (UE) knowledge, training, and experience in topics such as psychology, cognition, perception, and task analysis. The design and development of a functional core requires specialized software engineering (SE) knowledge, training, and experience in topics such as algorithms, data structures, and database management. Given that the user interface and the functional core are two closely coupled components of a system, and that each can constrain the design of the other, there should be close connections between the two development life cycle processes. Unfortunately, the two disciplines are practiced almost independently [3-5], which results in missed opportunities to collaborate, coordinate and communicate about the overall design and often leads to project failures [6-8].

To connect the SE and UE life cycles, we designed and developed the Ripple Implementation Framework [5, 9], within which developers can define the key components, knowledge base, tool support, timelines, and development environment necessary to support an interactive-system development effort. This framework embodies a storage sub-system containing shared design representations. Here, usability and software engineers can store, access, share, and manipulate the various work products that are created during a development effort. Another important component of this framework is the constraint and dependency sub-system that has capabilities to record, propagate, and enforce the various constraints that exist between the two life cycles and their resulting work products.

1.3 Evaluating Systems-Level Frameworks

Perhaps the most scientifically rigorous way to evaluate a software development framework such as the Ripple Implementation Framework would be to use a full, formal summative experimental design in which large number of software development teams (of the same size and balanced for skills and experience) are employed to develop the same software system, using the same software development methodology, for the same client, with half the teams randomly assigned to use an instantiation of the Ripple Implementation Framework and the other half not. By controlling all other factors that could potentially impact the quality of the system or performance of the teams in the experiment, one would hope that such an experiment would make it possible to establish cause-and-effect relationships between any quality or performance indicators of the process employed by each team and the resulting product due to the use (or not) of the Ripple Implementation Framework. Also, if the number of teams is large enough, one could check for statistical significance in the measures (or indicators) of dependent variables as a causal outcome of using (or not) the Ripple Implementation Framework.

However, such a summative evaluation of a software development framework such as Ripple with a large number of real teams of professionals building the same system

using different development frameworks is not practically possible. It is difficult and expensive enough for just one team to development any non-trivial software system. The most obvious reason is that such an undertaking would be ludicrously expensive given the resources necessary to conduct an experiment of such magnitude. Furthermore, given the complex nature of interactive-software development, it is impossible to control all factors (assuming it is possible to identify all factors in the first place) involved in the development process. Finally, even if somehow all these problems could be solved, any results from such an experiment will suffer from external validity problems as these results cannot necessarily be generalized to contexts where a different set of factors are at play: different sizes of development team, different skills or experience of teams, different software development methodologies in use, different type of software system being developed, different project management styles, different types of client interaction, etc.

1.4 Planning a Testbed Evaluation Environment

Given the impossibility of conducting a formal summative study to evaluate a framework like Ripple, we sought alternative approaches. After two years of planning with the Department of Computer Science at Virginia Tech, we used a joint offering of graduate-level SE and UE courses for this evaluation. Students from the UE class were trained for the usability engineer role and taught the life cycle concepts and guidelines for building user interfaces. Similarly, students from the SE class were prepared for the software engineer role and taught the life cycle concepts for functional core development.

Instantiating the Ripple Framework for a classroom setting we were able to test the framework concept before investing resources in its full software implementation. We achieved this with a kind of Wizard-of-Oz approach to simulate various aspects of the framework behind the scenes. For example, one of the authors played the role of various sub-systems in Ripple instead of actual software implementations. Email was used in place of automated Ripple messages and a password-protected, forum-based, and archived group email software was used as the shared design representation for developers to post their work products and share them with the other members in their teams.

1.5 Team Composition and Project Description

We had a total of eight teams each following one of four different models of software development (Ripple instantiation was one such model) as shown in Table 1. Semester-long projects were used to build a real-world software system for the Horticulture Club of Virginia Tech. Three members from the Horticulture club were compensated and trained to act as client representatives for all the teams. Special care (interactions with client representatives were monitored) was taken to prevent one team's interaction with the client from influencing their interaction with another team.

Table 1. Team setup and distribution

Team	Team type	SE role	UE role	Role/setup/end product
A1	Ripple	3 SE	3 UE	• Distinct roles working together • Knew counterpart team from start • Developed fully-functional system
A2		3 SE	4 UE	
B1	non-Ripple	3 SE	4 UE	• Distinct roles working independently • Counterpart team introduced post design • Developed fully-functional system
B2		3 SE	4 UE	
C	Dual experts	3 students who took both classes		• Played both roles • Developed fully-functional system
D1	UE-only team	-	5 UE	• Played usability engineer role only • Had no counterpart team • Developed hi-fidelity prototype only
D2		-	5 UE	
D3		-	4 UE	

2 Need for a Goal-Directed Approach to Evaluation

In the absence of a formal summative study, our objective became a semi-formal study, but we did not find much work in the literature about what metrics to use or how to arrive at them. Fundamentally, when setting up an evaluation, questions arise as to what to measure (i.e., metrics). In order to know this, we need to use a systematic approach to arrive at the questions that we need the data to answer. To decide that, one needs to clearly identify the goals of the evaluation.

One approach that embodies this goal-directed approach to derive metrics is the Goal-Question-Metric (GQM) paradigm, originally used in the software engineering domain [10, 11]. In this method one should first postulate the goals of evaluation and derive questions, the answers to which will indicate if the goals are met. Each of those questions is then analyzed in order to determine the measurements that are necessary to answer the question.

2.1 Applying GQM to the Ripple Framework

We started with the goal of this study, which pertained to the Ripple implementation instance:

Evaluate the effectiveness of the Ripple framework, as embodied in a specific implementation instance, to facilitate communication among developers within and between the two life cycles.

High-level questions, Q1 and Q2, can be derived from the goal:

Q1. At the highest level, what are the indicators appropriate to reflect effectiveness of framework?

Q2. To what end and in what ways is communication used in the Ripple framework?

Since an effective framework should result in better quality of process and resulting product, question Q1 can be answered as:

A1. The quality of product and process.

And, since communication in Ripple is used to facilitate various other factors, question Q2 can be answered as:

A2. Communication, the main contribution of the Ripple framework, is used to facilitate coordination, synchronization, constraint and dependency checking, and change management.

Now, one can ask, in questions Q3 and Q4, what measures are needed to provide these answers? The end result of this exercise is shown in Fig 1.

However, certain changes and adjustments had to be made to these metrics as a result of the various constraints inherent in our study. For example, it was not possible to conduct a lab-based usability evaluation of all teams' products to arrive at measures such as time on task, number of errors, and satisfaction. The reason for this is that three D teams developed high-fidelity prototypes only and did not have a functional backend. Each of these prototypes were constructed to simulate carefully scripted, but different, benchmark tasks only and therefore it was not possible to run the same set of tasks in a lab setting across all teams. Therefore we created a metric called overall value index based on product quality ratings by the client representatives (described below).

Similarly it was not possible to compare code size and complexity using an objective measure because the D teams had no functional core software. Also, in the five joint teams that created fully functional systems, they adopted different programming technologies ranging from AJAX to JSP, making it impossible to compare using code-based metrics.

2.2 Evaluation Instruments and Metrics Used

Using the approach described above, we derived the following evaluation instruments and associated metrics:

2.2.1 Value Index of a System

At the end of the semester, we conducted an eight-hour meeting in which, the experimenter and all three client representatives analyzed the prototypes for breadth of functionality covered by the UI, usability, and appropriateness for the Horticulture Club's goals. Each team's final system was analyzed in detail to arrive at an overall value index per team product. First, we compiled an exhaustive "union" list of all features across all teams (e.g. ability to search by common plant name, ability to view shopping cart at all times, feature to provide directions to plant sale, etc.). Each of these features was then ranked on a desirability scale, which indicated how important this feature was for the client's plant sale. The scale included a range of low, medium, high with fractional values in between in some cases. In other words this analysis provided a superset of all features with each feature's desirability present in all systems combined.

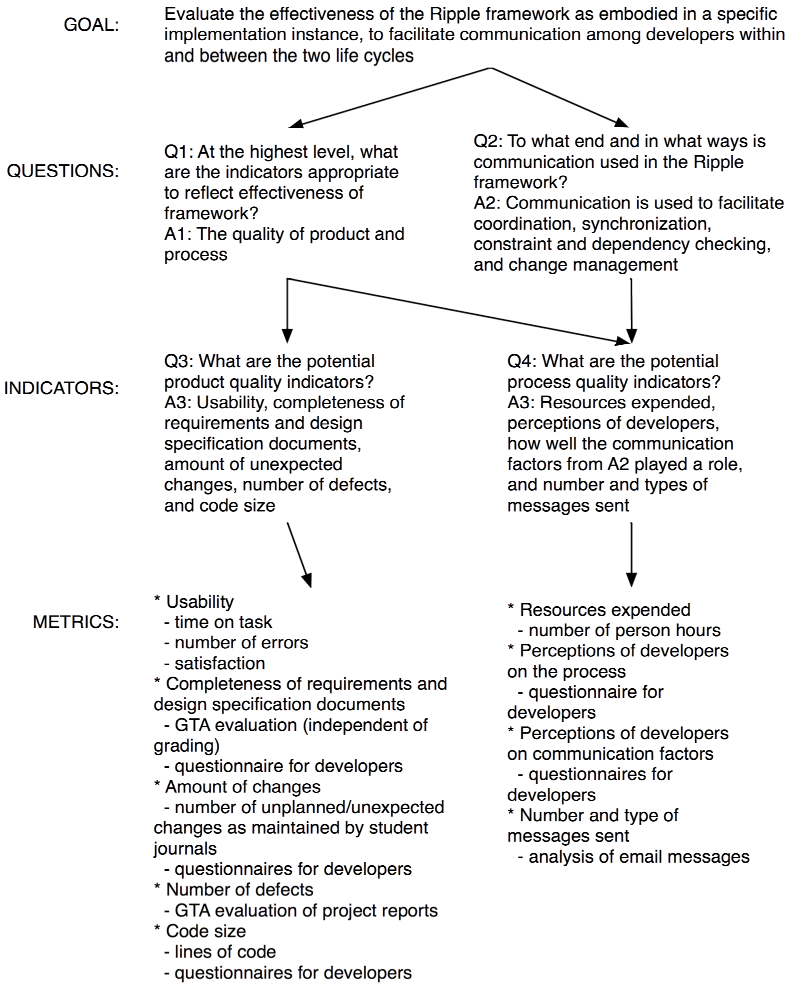

Fig. 1. GQM applied to Ripple to derive metrics

After this desirability analysis, the clients also analyzed each feature of each team's product, rating it on a perceived quality scale that included values of "poor", "fair", and "good" with fractional values in between for some cases. On a matrix, each feature on the union list was marked with a one or zero to indicate its presence or absence, respectively, combined with the ratings for desirability and perceived quality. A product of these three values (feature present/absent, desirability rating, and perceived quality rating) was computed to arrive at the value index per feature. This value index per feature was aggregated to calculate the total value index per team product. We also computed an aggregate feature count per team. These two values provided an overall comparison of the different systems by each team.

One of the issues faced in comparing the value of the systems developed by the different teams in the study was their dissimilarity in terms of the "realness" of the

systems. For example, the three D teams created systems without a real backend (i.e. user interface prototypes only). Therefore, the value indexes for D team systems were based on more perceived quality attributes than real ones. However, we believe this value index still serves as a basis of comparison within sets of teams that used the same development condition.

Even though the value index metric proved useful in rating the quality of different teams, one issue we noticed with this instrument was the fact that it did not provide an insight into the contribution of each sub-team in the case of A, B, and C teams. For example, a low value index indicates a failure on the part of the entire team but does not describe if the failure was due to low-quality contributed by the UE sub-team or the SE sub-team.

A potential further problem with this index is the fact that it is not tolerant to distortion due to feature creep. For example a team could accrue a high value index if its system has fewer core (essential) requirements but a large number of non-essential requirements, which would probably never be used by the client. This effect was somewhat offset by very low desirability scores for not-so-useful features, bringing down the overall feature value index. This same problem has the potential for a more pronounced effect in the second metric derived from this instrument: feature count per system. However, to our knowledge, this was not a severe problem in this analysis.

2.2.2 Activity Journal Analysis

As part of this study we required all students in both classes to maintain an online journal to record individual and group hours spent working on the project, problems encountered, strategies used, negotiations held, overall impressions of the process, and other project-related details. We analyzed the qualitative data in the journals (unexpected changes, issues with SE-UE interaction, and other experiences) in each entry of each developer's sheet and used that information as evidence to investigate and reconstruct what happened during this exploratory study. The activity journals proved to be an important source of information in the investigation of how the teams performed and aided in identifying a list of factors that seemed to play a role in the interactive-software development space. Because of the confidentiality of these journals, students were willing to share candidly their insights into what was working in the process, team problems, rationale for design decisions, etc. and provided explanations for some of the phenomenon we observed during the study.

2.2.3 Email Analysis

In order to measure the amount and nature of communication that transpired among the various members in each team, all teams were provided with custom group email IDs in which the experimenter was a silent member. The joint teams each had three email IDs: one for the SE group, one for the UE group, and one for the combined group. The students were required to use these group email IDs for all project-related communication during the semester. The entire collection of email exchanged among all groups throughout the semester was archived and analyzed. Each email was tagged with various keywords. The total number of emails per team, the frequencies of keywords, and some subjective analysis were used as metrics. This instrument provided deep insights into SE-UE sub-team dynamics. Using this instrument, we could identify the strategies different teams used, how they negotiated different problems in the

team, etc. This instrument combined with the activity journal entries provided a surprising amount of nuanced information about each team's operations, their overall performance, and their working relationships.

2.2.4 Transcripts of the End-of-Semester Symposium

Given the exploratory nature of the study, it was essential for us to understand each team's final perceptions of problems inherent in the development model they were assigned, the strategies they used to overcome these problems, lessons learned as part of their experience, and (based on these experiences) any advice they have for real-world software developers. In order to facilitate this kind of debriefing and sharing of knowledge we hosted a research symposium at the end of the semester in which each team presented the key findings from their project experience. This symposium was recorded (audio and video) and the content transcribed (text files). Components of data from the transcripts were used as metrics for the study. Even though this transcription exercise took significant effort, it provided rich data about what worked for each team and what challenges they faced. This instrument provided an overview of the entire process (as opposed to the value index providing overview of the product) in the project as perceived in retrospect by the teams.

2.2.5 Group Interviews

At the end of the semester (after the symposium) each team was individually interviewed for an hour and a half. The interview was semi-structured with general questions probing the challenges faced (gleaned from the end-of-semester symposium), how prior work or academic experience in each role affected their performance in the study, how their interaction with the counterpart team affected their experience, what style of development they preferred, and other impromptu questions resulting from their answers. The interviews were recorded (audio only) and transcribed. Various parts of the data from these transcripts were used as metrics for the study. Once again, this was a tedious and time consuming exercise, but one which yielded rich qualitative data about many aspects of each team's experiences.

2.2.6 Surveys

At the end of the group interviews, each student was administered two written surveys: one with 55 questions gauging their perceptions on a wide variety of aspects of the study and another with 22 questions gauging their perceptions on various pedagogical aspects of the joint offering of SE and UE courses. The data from these surveys were aggregated to team level responses and analyzed. Also, all students were given the same two surveys regardless of the development condition they used for their group project. Therefore, for questions aimed at gauging their perceptions about other development conditions, the students had to resort to conjecture based on their experience observing the other teams in class throughout the semester and during the symposium presentation. For example, some questions in the survey asked the students about their perceptions on the importance of having periodic communication between SE and UE roles in an interactive-software development effort. However, the D teams had no first-hand knowledge with this issue as they did not have a counterpart SE role. The surveys provided some statistically significant trends about broader issues like the importance of communication in teams.

2.2.7 Subjective Feedback from Clients

After each interaction the clients had with a team (for example after their requirements gathering meetings, formative evaluation meetings, etc.) and after the end-of-semester product comparison meeting, the clients were asked if they had any general impressions on the team's performance in that meeting or their overall impression on the team's system. The feedback from clients in these situations were recorded and used as metrics. These were purely subjective assessments by the clients that provided a perspective that was not captured in any of the other instruments described above. Whereas each of the other instruments (except, perhaps, the symposium) provided unique insights into a small subset of the aspects involved in this study, this client feedback provided a holistic, albeit general, assessment of each team. Their feedback represented those aspects that are more about the overall interaction and impression they had about each team and its system.

3 Outcomes and Conclusion

In the end this evaluation proved to be a success. The data from the different metrics provided insights into different aspects of the study. In many cases insights gained using one instrument were supported by others. For example, we found strong evidence showing the effectiveness and utility of Ripple-like communication-fostering frameworks (both quantitative survey trends and qualitative discourse from journals and interviews). We also discovered that certain aspects of social dynamics of collaborative work can outweigh the effects of structured communication or its absence (from email instrument we found one SE team ignore dozens of emails from UE counterparts about UI design specifications). We found evidence for inherent conflict of interest when the same people perform both SE and UE roles and often adopt design solutions that are easy to implement rather than implementing solutions that are easy to use (gathered from client perceptions, symposium transcripts, and via team interviews). Another interesting finding was regarding UE-only D teams. Their unconstrained development context (i.e. no software developers to negotiate feasibility and development constraints) resulted in broad and rich designs that turned out to be some of the client's favorite picks (evidenced by team interviews and journal entries where team members commented about not having to face the implementation challenges other types of team were facing).

In conclusion, in this paper we described a series of measuring instruments we adopted to investigate the effectiveness of the Ripple Implementation Instance and to explore the various communication factors that could potentially impact interactive-software development. Each of these instruments provided a unique insight into the investigation of each team's performance and to the general understanding of the different factors that seem to influence the quality of interactive-software development. However, no single instrument, by itself, provided irrefutable evidence to explain beyond reasonable doubt the various factors that were at play in this exploratory study. Using an analogy of the criminal justice system, combinations of empirical "evidence", ranging from objective to subjective and from qualitative to quantitative,

were pieced together to determine our understanding of what happened with each team in the study. Based on our experiences with this study we believe a broad goal-oriented and systematic approach to evaluation in general and metrics in particular provides an effective way to evaluate systems-level frameworks.

References

1. Hartson, H.R., Andre, T.S., Williges, R.C.: Criteria for evaluating usability evaluation methods. International Journal of Human-Computer Interaction 15(1), 145–181 (2003)
2. Monk, A.F.: Experiments are for small questions, not large ones like "What usability evaluation method should I use? Human-Computer Interaction 13(3), 199–201 (1998)
3. Pyla, P.S., et al.: Towards a model-based framework for integrating usability and software engineering life cycles. In: Interact 2003 Workshop on Closing the Gaps: Software Engineering and Human Computer Interaction. 2003: Université catholique de Louvain, Institut d' Administration et de Gestion (IAG) on behalf of the International Federation for Information Processing (IFIP), pp. 67–74 (2003)
4. Pyla, P.S., et al.: What we should teach, but don't: Proposal for a cross pollinated HCI-SE curriculum. In: Frontiers in Education (FIE) Conference, Savannah, Georgia, pp. S1H17–22 (2004)
5. Pyla, P.S., et al.: Ripple: An event driven design representation framework for integrating usability and software engineering life cycles. In: Seffah, A., Gulliksen, J., Desmarais, M. (eds.) Human-centered software engineering: Integrating usability in the software development lifecycle, pp. 245–265. Springer, Heidelberg (2005)
6. The Standish Group, The CHAOS Report (1994)
7. The Standish Group, Unfinished Voyages. A follow-up to The CHAOS Report (1995)
8. The Standish Group, Extreme CHAOS (2001)
9. Pyla, P.S., et al.: Evaluating Ripple: Experiences from a Cross Pollinated SE-UE Study. In: CHI 2007 Workshop on Increasing the Impact of Usability Work in Software Development, 4 pages (2007)
10. Basili, V.R., Weiss, D.: A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering SE-10(6), 728–738 (1984)
11. Fenton, N.E., Pfleeger, S.L.: Software metrics: A rigorous and practical approach, 2nd edn. International Thomson Computer Press (1997)