Published at PROFES 2001, pp 110 – 125

Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop

Victor Basili, Roseanne Tesoriero, Patricia Costa, Mikael Lindvall, Ioana Rus, Forrest Shull, and Marvin Zelkowitz

Fraunhofer Center for Experimental Software Engineering Maryland
{vbasili, rtesoriero, pcosta, mikli, irus, fshull,mzelkowitz}@fc-md.umd.edu

Abstract. New information is obtained by research and disseminated by papers in conferences and journals. The synthesis of knowledge depends upon social discourse among the experts in a given domain to discuss the impact of this new information. Meetings among such experts are, however, expensive and time consuming. In this paper we discuss the organization of CeBASE, a center whose goal is the collection and dissemination of empirically-based software engineering knowledge, and the concept of the online workshop or *eWorkshop* as a way to use the Internet to minimize the needs of face-to-face meetings. We discuss the design of our eWorkshop and give the results of one eWorkshop that discussed the impact of defect reduction strategies.

1 Building an Experience Base for Software Engineering

Software development is a people- and knowledge-intensive activity; it is a rapidly changing field, and although it is slowly maturing, many activities are still ad hoc and depend upon personal experiences. In order to cope with such restrictions as firm deadlines and shrinking budgets, software-developing organizations need assistance in setting up and running increasingly critical projects.

In order to reach their goals, software development teams need to understand and choose the right models and techniques to support their projects. They must answer key questions: what is the best life-cycle process model to choose for this particular project (from waterfall to extreme programming)? What is an appropriate balance of effort between inspections and testing in a specific context? And what are the benefits, if any, to buy a readily available software component instead of developing it?

These questions are not easy to answer. In some cases the knowledge exists to answer such questions; in other cases it does not, so instead on relying on knowledge and experience, we must trust our instincts. In order to support this decision-making activity, we need to develop empirically based software models in a systematic way, covering all aspects from high-level lifecycle models to low-level techniques, in which the effects of process decisions are well understood. However, context plays an important role as most projects and organizations differ. Consequently, the knowledge must be formulated relative to the development context and project goals.

The Center for Empirically-Based Software Engineering (CeBASE)[1] was organized to support this goal. CeBASE will accumulate empirical models in order to provide validated guidelines for selecting techniques and models, recommend areas for research, and support software engineering education. CeBASE's objective is to transform software engineering from a fad-based practice to an engineering-based discipline in which development processes are selected based on what is known about their effects on products, through synthesis, derivation, organization, and dissemination of empirical knowledge on software development and evolution phenomenology.

CeBASE is a new National Science Foundation-sponsored research center led by personnel with extensive industry and government experience, including Professors Barry Boehm (University of Southern California), Scott Henninger (University of Nebraska Lincoln) Rayford Vaughn (Mississippi State University) and co-author Victor Basili (University of Maryland and Fraunhofer Center for Experimental Software Engineering – Maryland). Their experience includes major improvements in software productivity, quality, and cycle time on highly complex software-intensive systems. Within CeBASE we are integrating existing data, methods, and models (including the Experience Factory, Goal-Question-Metric approach, Spiral life cycle Model, Model-Based (System) Architecting and Software Engineering (MBASE), stakeholder Win-Win requirements engineering, COCOMO cost and schedule modeling, and perspective-based reviewing) with the best commercial and government practices (such as the CMMI, Rational Unified Process, Independent Expert Reviews/Architecture Review Boards, SEI Product Line, Risk, and Architecture practices). The result will be tailorable methods for defining, managing, and continuously improving future software intensive systems involving challenges such as COTS based development, systems distribution and mobility, and tradeoffs among cost, schedule, performance, dependability, interoperability, usability, and other factors. The initial focus of CeBASE is on two high-leverage areas of software engineering, defect reduction and COTS based development.

CeBASE collects, documents, and disseminates knowledge on software engineering gained from experiments, case studies, observations, and real world projects. While some of this empirical knowledge might be well known by the community, it has not

[1] http://www.CeBASE.org

yet been documented. Although this knowledge is believed to be generally applicable, the effects of its application have never been systematically investigated making it difficult to discern when it is useful. Some of this knowledge is distributed among many individuals, which means that we need to gather the pieces together and facilitate the collection and management of collective knowledge.

Meetings among experts discussing their findings and recording their discussions are a classical method for creating and disseminating knowledge. By analyzing such discussions new knowledge can be created and the results can be shared. This is generally achieved by holding workshops. Workshops, however, possess limitations: 1) experts are spread all over the world and would have to travel, and 2) workshops are usually oral presentations and discussions, which are generally not captured for further analysis. To overcome these problems we designed the concept of the *eWorkshop*, using the facilities of the Internet. This paper describes the process of running an eWorkshop and the results from the first eWorkshop, which was held March 16, 2001.

2 Conducting an eWorkshop

The eWorkshop is an on-line meeting, which replaces the usual face-to-face workshop. While it uses a Web-based chat-application, it is structured to accommodate the needs of a workshop without becoming an unconstrained on-line chat discussion. The goal is to synthesize new knowledge from a group of experts as an efficient and inexpensive method in order to populate the CeBASE experience base. The idea behind the eWorkshop was to use simple collaboration tools, thus minimizing potential technical problems and decreasing the time it would take to learn the tools. Simultaneously, we wanted to set up a support team and control room to ensure that there would be as few disturbances as possible once the eWorkshop was running.

2.1 eWorkshop Process

Organization of the workshop follows a strict protocol:

1. *Choose a topic of discussion*. The topic under discussion is first determined.
2. *Invite participants*. Participants are invited and instructed to log into the eWorkshop using a Web browser at the appointed time.
3. *Distribute Pre-meeting information sheet*. To direct the discussion, preliminary information about the topic is presented to the participants, who send in a pre-meeting information sheet. This is used to guide the discussion during the meeting.
4. *Establish meeting codes – for meeting analysis*. The workshop organizers analyze the information sheets to develop a taxonomy of issues to be discussed.
5. *Publish synthesized info from pre-meeting sheets*. An analysis of the information sheets are given by the eWorkshop team and distributed to each participant before the meeting.

6. *Schedule pre-meeting training on tools*. A preliminary work session is scheduled to give meeting participants a chance to try out the software so that the meeting can proceed smoothly.
7. *Set up control room*. Several individuals (described later) actually run the meeting. While most participants are in their own offices looking at a computer screen, the meeting organizers need to coordinate their activities among several roles.
8. *Conduct meeting*. At the appointed time, the participants use their Web browser to log into the chat tool and the meeting is underway.
9. *Post-meeting analysis and synthesis*. The meeting organizers keep a script of the meeting. This is analyzed to extract knowledge for the knowledge base.

2.2 eWorkshop Roles

Most participants in an eWorkshop are experts in their respective domain. Our lead discussants (workshop leaders) formed part of the CeBASE team that interacted with an international group of invited participant experts. To minimize disturbances during the meeting and to capture important information, we relied on a support team operating from a single control room. This support team consisted of the following roles: *moderator*, *director*, *scribe*, *tech support*, and *analyst*. A phone line was set up in the control room and the number along with an email address were given to each participant to use if they had any technical problems during the meeting.

The moderator was responsible for monitoring and focusing the discussion (e.g., proposing items on which to vote) and maintaining the agenda. Of the support team, only the moderator was an active participant in the sense that he contributed actual responses during the meeting. The director was responsible for assessing and setting the pace of the discussion. He decided when it was time to redirect the discussion onto another topic. As the discussion moved from one topic to another, the scribe highlighted the current agenda item and captured and organized the results displayed on the whiteboard area of the screen. When the participants reached a consensus on a particular item through a vote, the scribe summarized and updated the whiteboard to reflect the outcome. The contents of the whiteboard became the first draft of the meeting minutes. The analyst coded the responses according to the pre-defined taxonomy. The analyst entered one or more codes to categorize responses as they were entered. The tech support was responsible for handling any problems that might occur with the tools. For example, some participants accidentally closed their sessions and had difficulty logging into the meeting for a second time. The tech support assisted these participants in troubleshooting their problems.

2.3 eWorkshop Tool

The web-based chat-application used to run the eWorkshop allows participants to create identities based upon their names. Instead of communicating statements orally, the participants submit statements by typing them and pressing the submit button. The statement appears on all participants' message boards. By responding to statements on the message board, participants can carry on a discussion online. This allows participants to be located remotely from the control room. Moreover, all statements are automatically captured in real-time, allowing them to be analyzed afterward.

The eWorkshop tool is based on web technology. The screen has five main areas: the *agenda*, *input panel*, *message board*, *whiteboard*, and *attendee list*. There are also features supporting the collaboration, such as a chat log and directory of frequently asked questions. (See **Figure 1**.)

The agenda is managed by the moderator and indicates the status of the meeting ("started", "stopped") as well as the current item under discussion. Immediately above the agenda is a link to the pre-meeting information submitted by the participants, which is used to establish a common understanding of the participants' status before the meeting is started. This is accessible throughout the meeting.

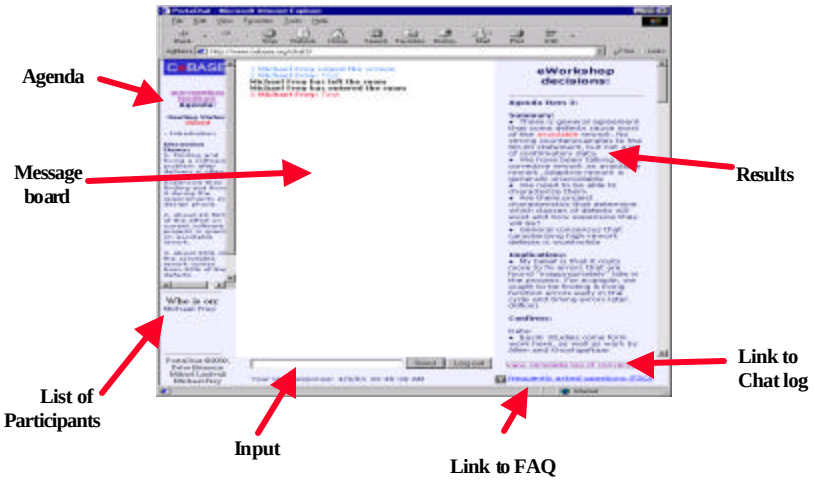

Figure 1 Screen shot of the chat tool used for the eWorkshop

The input panel enables participants to type statements during the discussion. The statement appears in the text box until the *Send* button is pressed.

The message board forms the meeting discussion area. Each statement is tagged with a unique number in addition to the name of the participant who entered it. Statements are displayed sequentially. The number is used for later referral to a particular statement. The statement is also tagged with the time of when it was sent (this piece of information is not shown in this panel, but in the chat log).

The whiteboard is used to synthesize a summary of the discussion and is controlled by the scribe. In order to realize our goal of measuring the level of consensus among the participants, all of the items added to the whiteboard were subject to voting announced by the moderator. When participants did not agree with how the statements on the whiteboard were formulated negotiations were initiated in order to come up with a more accurate description of the results of the discussion.

3 Defect Reduction eWorkshop

The first eWorkshop discussed defect reduction techniques and was based on an article [2] that provided a top-ten list of empirical data that many software practitioners found very helpful. An example of a top 10 defect item is: "Finding and fixing defects after delivery is 100 times more expensive than finding and fixing during requirement and design phase." This is an example of common knowledge that few people dispute in general, but it raises the question: "Does this knowledge always apply to all projects and under all circumstances?" In order to gain more knowledge, we decided to run eWorkshops with recognized experts on defect reduction from all over the world and collect data that supported or refuted the heuristics.

Based on this top-10 defect reduction list the goals of the eWorkshop were the following:

1. Gain a better shared understanding of key software defect reduction techniques.
2. Build a broader base of empirical results on software defect reduction to help support project decisions.
3. Conduct an assessment of the current eWorkshop's effectiveness in strengthening empirical software engineering knowledge.

3.1 eWorkshop preparations

Several activities based on the goals of the meeting took place prior to the actual eWorkshop. As described in Section 2.1, participants were asked to fill out and submit a pre-meeting information sheet regarding their views on the heuristics under discussion. To support the final meeting goal, we provided mechanisms to facilitate the use of the tools and established a measurement plan for evaluating the eWorkshop concept.

Content-related preparations

After potential participants responded to our invitation, we sent them the list of heuristics to be discussed during the eWorkshop and asked each to consider:

- If they had data that confirmed/refuted the heuristic;
- If they could refine the heuristic (e.g. by suggesting more accurate guidelines or specifying to what types of systems it applies);
- If they had other references or citations that were relevant as resources concerning this heuristic;
- If they could state the implications for practice, if this heuristic is true.

It was useful to group answers based on whether they supported or refuted the heuristic, to get a first impression of the amount of support on both sides of the issue. As in the actual eWorkshop, participants substantiated their answers with a range of different types of support, ranging from anecdotes to qualitative personal experience to references or hard data collected from a number of organizations. Rather than categorize the responses based on the level of support, we merely grouped responses on different sides of the issue and let readers draw their own conclusions about the relative merits of each.

The aggregated responses [2] were sent to all participants to allow them to familiarize themselves with the other participants, their backgrounds, and opinions before the actual chat. The aggregated responses were also useful during the chat, as they allowed participants to refer to data they had previously submitted without re-entering them during the real-time discussion.

Tool preparations

We asked all participants to test the meeting software to become familiar with the technology via two training sessions several days before the meeting. During that time, members of the support staff were present in the control room to answer questions and help the participants become more familiar with the tool. In addition, a list of frequently asked questions (FAQ) was placed on the CeBASE web site to explain how to use the software and identify conventions established to facilitate the discussion. Meeting participants were encouraged to read the FAQ and add additional questions, if necessary.

In order to evaluate the content of the meeting as well as the effectiveness of the eWorkshop as a forum for discussing software engineering topics, we established a data collection plan. Since the workshop was to be conducted using a chat-based tool that required participants to type each response, the actual transcript of the meeting was recorded. Additional user information such as the time of response, user name and IP address were captured and stored along with the text of each response. To obtain a better understanding of the meeting, we planned to code each participant's response according to a pre-defined set of categories based on the heuristics that

[2] Available at http://www.cebase.org/defectreduction/eworkshop1/premeeting.htm.

were to be discussed in the meeting (content-based categories) and on aspects of the meeting itself (meeting-based categories).

Post meeting analysis

In evaluating the eWorkshop concept, we are interested in two main points: the effectiveness of the concept in generating new information on the topic of defect reduction and the effectiveness of the tools in supporting the meeting. During the meeting, we collected data in various forms to enable an evaluation of these points. The following list represents the data sources available for the evaluation of the eWorkshop:

- the transcript of the actual text from the meeting
- the scribe summary approved by the participants
- the analyst's coding of each captured response
- the users and IP addresses by session

3.2 eWorkshop Concept Knowledge

The duration of the first eWorkshop was 2 hours. During that time, 533 responses were recorded. The meeting was globally distributed with participants joining the discussion from various parts of the United States (including Maryland, Pennsylvania, Mississippi, California and Hawaii), Europe (Denmark) and Asia (Japan). The connection speeds for the participants varied as well. Several of the participants attended the meeting connected to the Internet by a phone line using a slow modem while others had higher speed connections. The varying connection speeds did not seem to affect the ability of the participants to communicate. Figure 2 shows the distribution of the responses among the participants in the discussion during the meeting. While the lead discussants (Barry Boehm and Vic Basili) and the moderator contributed the most responses during the meeting (14%, 8%, and 11% of the total responses, respectively), 12 of the discussants contributed at least 4% (each) of the total number of responses.

Published at PROFES 2001, pp 110 – 125

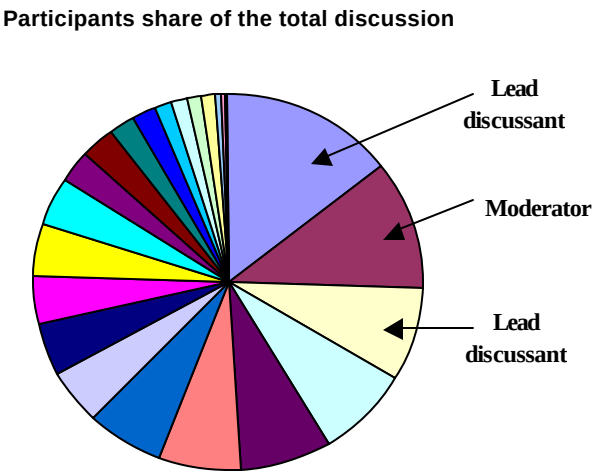

Figure 2: Each slice of the pie represents each participant's portion of the total discussion. The two lead discussants' and the moderator's slices are marked. A majority of the meeting participants were able to add to the discussion. There was no overwhelmingly dominant voice.

The analyst's coding of the responses yielded a characterization of the content of the meeting. Most of the responses submitted by the participants were content-related. Nearly 19% of the total responses during the discussion were related to voting. 17% of the responses dealt with rework effort. 13% of the responses focused on definitions. 12% of the responses discussed actual data and citations. During the meeting, 11 citations to references in the literature regarding software defect reduction were mentioned. Approximately 10% of the discussion dealt with procedural or administrative details. This amount of overhead seems to be consistent with face-to-face meetings. For example, in [9], a study of communication effort during inspection meetings was conducted. In the observed meetings (with average duration of 62 minutes), 14% of the communication effort was spent on administrative details.

In conducting the eWorkshop, we learned some lessons on conducting a meeting using this format:

1. There was no monopolizing voice during the meeting (unlike face-to-face meetings [11]).
2. Most of the discussion was content-related.
3. Participants need to prepare more for the meeting. Not many of the participants completed the pre-meeting information sheet and participants admitted to not preparing enough for the meeting.

4. It is important for participants to become familiar with the tool before the actual meeting.
5. The meeting seemed to run smoothly, but we did have some technical problems.
6. The simple, yet robust collaboration tools, such as the chat tool, seem to be adequate.

3.3 Defect Reduction Knowledge

In this section, we discuss the knowledge building that occurred concerning the software development heuristics that were the topic of the discussion. In this context, knowledge building generally took the form of validating whether the described heuristic held for at least some software systems; refining the heuristic by limiting the context in which it held; and suggesting implications for developers and researchers based on what is known and what data is missing.

Participants in this discussion included representatives from education:

- Ed Allen, Mississippi State University
- Victor Basili, Fraunhofer Center - Maryland and University of Maryland
- Barry Boehm, Center for Software Engineering, University of Southern California
- Winsor Brown, Center for Software Engineering, University of Southern California
- Philip Johnson, University of Hawaii
- Dan Port, Center for Software Engineering, University of Southern California
- Marvin Zelkowitz, Fraunhofer Center - Maryland and University of Maryland

from industry:

- Sunita Chulani, IBM
- Noopur Davis, Davis Systems
- Ira Forman, IBM
- Yoshihiro Matsumoto, Toshiba Software Factory, Japan
- Gary Thomas, Raytheon

and independent consultants:

- Don O'Neill, Don O'Neill Consulting
- Stan Rifkin, Masters Systems
- Otto Vinter, Independent Software Engineering Mentor, Denmark

Item 1

The first heuristic addressed the belief that finding defects early in development produces a large net savings in effort: “Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.” The discussion helped refine this heuristic (it is approximately right, for certain defect classes) and identified environments where it does not apply, or where there isn’t enough data to support a judgment.

eWorkshop participants generally agreed that finding and fixing software defects after delivery is much more expensive than fixing them during early stages of development – but only for certain types of defects. A 100:1 increase in effort from early phases to post-delivery was a usable heuristic for severe defects, but for non-severe defects the effort increase was not nearly as large. This heuristic is appropriate only for certain development models: new paradigms such as extreme programming (XP) remove any kind of meaningful distinction between "early" and "late" development phases, and research has not targeted such environments yet.

General data were presented that supported an effort increase of approximately 100:1. Don O'Neill described data from IBM Rochester [6] in the pre-meeting feedback that reported an increase in effort of about 13:1 for defect slippage from code to test and a further 9:1 increase for slippage from test to field (so, a ratio of about 117:1 from code to field). Based on Yoshihiro Matsumoto's background in a software factory of 2600 IT workers, average rework time after shipment is 22.85 hours versus less than 10 minutes if the work had been done prior to shipment (a factor of 137:1). Other corroboration came from Ed Allen's experiences with a telecommunications client as well as Noopur Davis' experience.

An important distinction that emerged was that the large effort multiplier holds for *severe* defects; fixing defects with lesser impact after delivery will not increase costs appreciably.

- Barry Boehm pointed out that the 100:1 factor was about right for critical defects on large projects, illustrated, for example, by the data from the Data Analysis Center for Software [7].
- Sunita Chulani also agreed that this factor was consistent with her experience for severe defects.

For non-severe defects, the effort multiplier was not nearly as large. Otto Vinter indicated that when requirements defects are excluded, his data show an approximately 2:1 relationship between after-shipment and before-shipment debugging effort: 14 hours after release versus 7.4 hours in testing before release. Barry Boehm said that the 2:1 relationship also held for the million-line CCPDS-R project completed by TRW for the Air Force (described by Walker Royce [8]), in which early risk resolution and well-validated modular architecting were used to reduce early defects. Marvin Zelkowitz also provided development data from NASA's Johnson Space Center, which showed that the effort multipliers associated with different defect types are different: the effort just to *find* a defect increased from

- 1.2 hours early in the project to 1.5 hours late in the project, for non-severe defects
- 1.4 hours early in the project to 3.0 hours late in the project, for severe defects.

Other variables likely to have an impact were proposed, although no supporting data was available. Gary Thomas pointed out that post-shipment costs would be expected to increase even further when a different organization than the one that developed the

software maintains the system. Winsor Brown suggested that the ratio would be different for the use of a ROTS (Research Off the Shelf) system. Since such software is offered "as is", the effort spent on finding and fixing defects is minimal, and when defects *are* found and fixed, the cost is usually minimal (since only the easy ones get "fixed"). And Philip Johnson remarked that research has so far neglected development environments that do not fit into the "waterfall family" of development approaches. For example, in XP, requirements and implementation phases are so entwined that it no longer makes sense to talk about "early" vs. "late" phases of development.

Item 2

The second heuristic concerned the impact of defects on project effort: "About 40-50% of the effort on current software projects is spent on avoidable rework." Again, discussants believed this heuristic to be about right for certain types of projects – however, they helped refine the heuristic by suggesting typical rework rates for different classes of projects.

Most eWorkshop participants believed that significant amounts of effort are spent on avoidable rework. The data across many projects had however a much wider range than the proposed 40-50%; on some projects cited, for example, it was as low as 10-20%. In general, it was felt necessary to distinguish different types of software engineering processes so that we could examine the avoidable rework rates for different types of environments. Several implications for addressing high rates of rework were suggested, but the one that attracted the most agreement argued that we need to collect more data to demonstrate these problems better: when data are collected the cost of defects and the benefits of removing them early are suddenly very evident.

Several participants cited data in support of large amounts of rework on projects:

- Vic Basili said that the 40-50% claim is borne out by the Cleanroom studies at NASA Goddard's Space Flight Center [1]
- Barry Boehm pointed out that Capers Jones' books [5] have data on rework costs that indicate that the rework fraction increases with the size of the project, and can go as high as 60% for very large projects.
- Don O'Neill submitted data from the national benchmarking effort showing that the range across projects is between 20% and 80%.

There was some agreement however that higher-maturity projects spend considerably less effort on rework. Brad Clark [4] has published analyses of the effects of process maturity in which the benefits at higher levels of maturity are traced mainly to the reduced rework effort.

- Gary Thomas, using data from Raytheon, cited a range of about 10-20% avoidable rework on higher-maturity projects.
- Barry Boehm said that the better TRW projects, such as CCPDS-R, were also able to reduce rework effort down to 10-20%.

Because of this disparity between high- and low-maturity projects, Don O'Neill suggested that we should distinguish among *disciplined software engineering*, *structured software engineering*, and *ad hoc programming* and seek to associate with each a characteristic level of effort spent on avoidable rework.

In general, comparing rework costs across projects is dangerous because it can be defined in several different ways. (For one example, Vic Basili pointed out that the rework effort collected from the Software Engineering Laboratory (SEL) at NASA was measured as the effort required to make changes due to defect corrections.)

Yet there are other potential rework measures for which researchers might not even be able to collect metrics, for example the rework that is found on volatile development teams, where people are often added or removed and as a consequence spend time relearning or redoing the same things.

Some implications of high rework effort were noted. Stan Rifkin proposed that for the present, we have to expect high rework effort and use that to help projects budget effort better. Stan Rifkin, Vic Basili, and Noopur Davis concurred that collecting data is a necessity, since once people start tracking defects, the cost of defects and the benefits of early removal become very clear.

Otto Vinter proposed that the research community focus more on studying the causes of defects and finding preventions for them. Noopur Davis took this argument a step further by saying that defect prevention efforts should concentrate on removing defects as early in the lifecycle as possible; Gary Thomas suggested that formal inspections help in this regard [10]. Barry Boehm reported that at TRW it was found that early prevention effort (via reviews, inspections, and analysis tools) had a 5:1 or 10:1 payoff.

Item 3

The third heuristic concerned the contribution of various defects to the amount of rework on a project: "About 80% of the avoidable rework comes from 20% of the defects." Unlike the earlier heuristics, little existing data addressed this topic.

There was general agreement that relatively few defects tend to cause most of the *avoidable* work on software projects. (However, it was clear that there is a significant amount of additional unavoidable rework that comes from sources such as adaptive maintenance. Discussants felt that software engineering researchers need to spend more work on characterizing the types of rework and their causes.) There was little confirmatory evidence about the 80/20 rule, but there were no strong counterexamples either. In terms of the implications of this statement, there was a general consensus that characterizing high-rework defects would be worthwhile.

On this topic, little data was put forward. While most participants indicated that they believed that most of the avoidable rework comes from a small number of defects, no

data from personal experience was cited. Some confirmatory data from the SEL and from the work of Khoshgoftaar and Allen was described. A dissenting voice came from Noopur Davis, who felt that the sheer number of defects that have to be found and fixed must contribute more than 80% of the avoidable rework.

Some time was spent trying to make definitions clearer. First, "rework" was defined broadly to include the effects of such things as changing operating systems, databases, or customer base; possibly also the re-configuration of tools.

Finding a broad definition of "defect" was more difficult. Winsor Brown said that this is not a unique problem during the eWorkshop: for example, in the absence of objective definitions the ODC (Orthogonal Defect Classification) researchers were forced to define a defect broadly, as any change made to the software. Stan Rifkin suggested that the CAPPU taxonomy used in the software maintenance community could help refine the definition. CAPPU characterizes changes to systems as Corrective (to mitigate the effect of errors), Adaptive (to port the system), Performance-related (to improve system performance), Preventive (to change something before it causes a problem), and User-requested (to add new functionality). The "defects" referred to in the 80/20 rule refer specifically to changes in the corrective and performance-related categories only.

Some definitions of "unavoidable rework" were presented to allow participants to agree on the types of things that were *not* covered by the agenda item. The basic idea, which seemed to have consensus, was that unavoidable rework was rework that came from other sources than defects, e.g. from Adaptive, Preventive, or User-requested changes to the code or architecture. Otto Vinter suggested expanding this definition by proposing that unavoidable rework could be caused by some defects that are simply too hard to prevent.

Most of the discussion centered on identifying what types of defects were most likely to cause disproportionately large amounts of rework. Barry Boehm said that in his experience one source of high-rework defects is "architecture-breakers:" defects whose fix requires significantly changes to the architecture, which then necessitates design and code changes.

Stan Rifkin described his belief that it costs more to fix errors that are found "inappropriately" late in the process. For example, we ought to be finding and fixing function errors early in the cycle and timing errors later, so function errors that are not found until the later stages will not cause a higher amount of rework. Barry Boehm commented that IBM ODC data (from Ram Chillarege's papers) [3] do indicate that some defects tend to be found earlier (e.g., function defects) and others tend to be found later (e.g., timing defects). Another implication of this is that timing defects are probably more expensive to fix than function defects, because they cannot be found in earlier phases where corrections would be cheaper.

Winsor Brown and Otto Vinter suggested another potential classification of defects: classifying based on potential prevention technique. Clearly some defects could have been avoided with better education, training, or information.

3.4 Participants' Feedback

The majority of the participants liked the eWorkshop. They thought it was a "good way to discuss," "worthwhile and stimulating" and "a relatively easy way for a group to get a discussion going." Some of the difficulties people reported were related to the tool, but most resulted from their lack of preparation in using the technology before the meeting.

People generally said they would participate again and would recommend this discussion vehicle to others, with the comment that more pre-meeting preparation would be a great benefit. More thorough preparations could include sending their position relative to the topics on the agenda, together with arguments to support or refute them, such as data and references.

4 Next Step

The first eWorkshop addressed the first three items of the top-ten list that focused on cost and effort. The second eWorkshop will be held on July 16, 2001 at 11:30 AM EDT. It will focus on the impact that defects have on software. These correspond to top-10 heuristics 4, 5, 9 and 10. A third meeting will discuss the remaining top-ten list items on effective methods to find defects. After the defect reduction eWorkshops, a new series of eWorkshops will be held that will discuss the top-ten list for COTS based systems.

We are currently improving the chat tool based on suggestions from the first eWorkshop and this improved version will be used for the second eWorkshop.

Although the eWorkshop idea proved so far to be an efficient approach for gathering collective knowledge, we do not believe the technology has progressed to the point where all face-to-face meetings can be eliminated. For this reason, we are planning to have a (no "e") workshop to conclude this eWorkshop series in Spring 2002.

We are constantly updating our website, its support tools and feedback mechanisms, to facilitate interaction between CeBASE experts and the software engineering community. Please visit our website at http://www.cebase.org frequently for more information!

5 Conclusions

CeBASE has an ambitious goal of collecting relevant empirically-based software engineering knowledge. The eWorkshop has been proposed as a mechanism for inexpensively and efficiently capturing this information. Based upon an initial virtual meeting in March 2001, the process seems effective, and most participants were satisfied with the results. We have learned from some flaws in this initial process, however, and are evolving the concept to where we believe it can replace a significant portion of face-to-face meetings using the virtual meeting capabilities of the Internet. We are planning several more meetings over the next few months where this concept can evolve.

The results from the first eWorkshop generally confirm the first three items in the top-ten defect reduction list. We obtained additional references and data that seek to classify when specific defect reduction heuristics are applicable. We are in the process of further analysis of this data, which we will place on the CeBASE web site. Once we refine our eWorkshop capabilities, we can run eWorkshops more frequently on a wider variety of topics to help evolve the CeBASE experience base rapidly.

6 Acknowledgments

This work is partially sponsored by NSF grant CCR0086078 to the University of Maryland with subcontract to the Fraunhofer Center - Maryland.

We want to thank Barry Boehm, Scott Henninger, Rayford Vaughn, Winsor Brown, Dan Port and Michael Frey as well as all participants for their contribution to the success of the eWorkshop. We also want to thank students at USC and UMD for their contribution in testing the system and Jennifer Dix for proof reading this paper.

References

1. Basili, Victor R and Green, Scott, "Software Process Evolution at the SEL," IEEE Software, pp 58-66, July 1994.
2. Boehm, Barry and Basili, Victor, "Top 10 Defect Reduction Techniques," IEEE Computer, January 2001. (Also available at http://www.cebase.org/defectreduction/top10/.)
3. Chillarege, Ram, Bhandari, Inderpal, Chaar, Jarir, Halliday, Michael, Moebus, Diane, Ray, Bonnie and Wong, Man-Yuen, "Orthogonal defect classification – a concept for in-process measurements," IEEE Trans on Software Engineering, 18(11), November 1992: 943-956.
4. Clark, Bradford, "The Effects of Process Maturity on Software Development Effort," Ph.D. Dissertation, USC, 1997. (http://sunset.usc.edu/reports)

Published at PROFES 2001, pp 110 – 125

5. Jones, Capers, Applied Software Measurement, 1996.
6. Lindner, Richard J. & Tudahl, D. "Software Development at a Baldrige Winner," Proceedings of ELECTRO `94, Boston, Massachusetts, May 12, 1994, pp. 167-180.
7. McGibbon, Thomas, Software Reliability Data Summary, DACS, 1996.
8. Royce Walker, *Software Project Management: A Unified Framework*, The Addison-Wesley Object Technology Series, 1998, Appendix D.
9. Seaman C. B. and Basili V.R., "Communication and Organization: An Empirical Study of Discussion in Inspection Meetings," IEEE Transactions on Software Engineering, 24(6), June 1998.
10. Don O'Neill, "Peer reviews and software inspections," Encyclopedia of Software Engineering, Wiley Publishing. To be published. Draft can be found at http://members.aol.com/ONeillDon2/peer-reviews.html.
11. "Fundamental Skills Course" GroupSystems.com, 1999