Software development is a highly abstract process that requires intense concentration. The authors show that interrupting this process can significantly reduce a developer's efficiency and can even contribute to project delays.

Interrupts: Just a Minute Never Is

Rini van Solingen, Schlumberger Retail Petroleum Systems
Egon Berghout, Delft University of Technology
Frank van Latum, Dräger Medical Technology

Software development consists of more than writing programming code. Developers also participate in discussions, meetings, phone calls, and activities undertaken to share knowledge and experience such as consulting on previously released products, explaining technical details, and reviewing documents. Surprisingly, software engineering research hardly addresses this issue, instead emphasizing software development methods and techniques. If we assume that software developers spend more than half their time communicating, we can legitimately question whether software engineering research has the right focus.

Norman Fenton and Shari Pfleeger support this focus on human factors when they state that "For many years, sociologists have studied personnel attributes, both as individuals and teams, and their effect on productivity and products. These characteristics include age, level and type of education, intelligence, gender, marital status, type of remuneration, and more. Although we do not yet measure these aspects of software developers and their work habits, it is clear that when researchers find them relevant to other professions, they are likely to be relevant to ours."[1]

0740-7459/98/$10.00 © 1998 IEEE

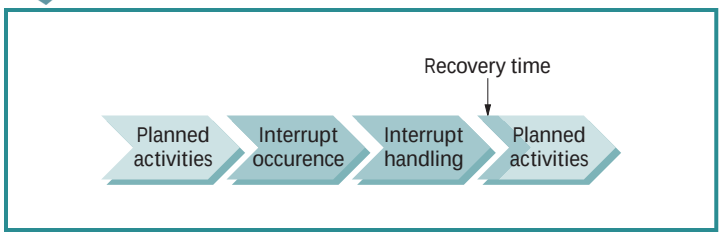

Figure 1. The three phases of an interrupt show that not only do the occurrence and handling of the interrupt subtract from the time devoted to planned activities, but developers lose further time recovering their concentration.

Tom DeMarco and Tim Lister give further evidence that human factors are important to software engineering. They suggest several ways to improve team interaction and trust, including nontraditional events such as gathering to cook a spaghetti dinner.[2] Bob Glass, noting software engineering's creative and heuristic nature, asserts that informal approaches might work better than formal bureaucratic ones.[3] He warns against the bureaucratic and noncreative effects that formal approaches and procedures tend to invoke. Even the highly process-oriented Software Engineering Institute has initiated a people-oriented approach to software engineering with the maturity reference model of human factors they've defined. The SEI applied their popular five-level CMM model to describe the maturity with which an organization addresses the importance of people: the People-CMM (P-CMM).[4] Watts Humphrey recently developed an approach for individual improvement of software engineers, the Personal Software Process, that also places greater emphasis on human factors.[5]

Armed with the conviction that human factors are an important contributor to software developers' effectiveness, we researched one specific human factor: the effects of interrupting developers during their work. The software development literature contains little research on interrupts. Dewayne Perry and colleagues report that software developers spend, on average, 15 percent of their daily effort on "unplanned interpersonal interaction."[6] DeMarco and Lister describe the high impact of phone calls in engineering environments: developers routinely receive 15 telephone calls a day, which can make the whole day nonproductive.[2] This signals that interrupts are a relevant issue for software development, and that they can be quite expensive. For example, consider the productivity loss implicit in an interrupt that occurs during a meeting and puts a discussion of eight people on hold for 15 minutes.

Interrupts Defined

For our research, we define an interrupt as any distraction that makes a developer stop his planned activity to respond to the interrupt's initiator. We further divide an interrupt into three phases:

1. *Occurrence.* An interrupt occurrence makes a developer stop his or her planned activities: the telephone rings, an important e-mail arrives, or a manager pays a visit. We include in this phase the effort needed to fully understand the interrupt.
2. *Handling.* The developer handles the interrupt, which implies that he or she works on the interrupt until the initiator is satisfied with the result. Developers usually handle interrupts immediately after they occur, but can sometimes postpone handling them until later.
3. *Recovery.* The developer resumes his or her planned activities. Developers must spend some time returning to the point in their work at which they were interrupted. We refer to this as recovery time. Although this time is spent on planned activities, it is an immediate interrupt effect.

Figure 1 shows these three phases. The boxed text "An Interrupt Example" on page 99 describes a typical interrupt in detail.

Measuring Interrupts: Two Cases

We studied interrupts in two organizations that develop embedded systems: Dräger Medical Technology and Schlumberger Retail Petroleum Systems. In both organizations, the development groups experienced problems with interrupts and commissioned a study to better understand them. The companies expected that a basic understanding of interrupts' frequency and causes would provide the data for interrupt planning, causal analysis, and corrective actions.

We encountered surprisingly similar results in both organizations, and even more remarkably, some results—such as interrupts taking 15-20 minutes each and 15-20 percent effort spent on interrupts—are exactly similar to what little data is available in software development literature. Our study on interrupts applies the generally accepted Goal/Question/Metric method[7,8] to address the in-

terrupt problem and to structure our study. Using GQM for this purpose is quite unusual, because we apply what is generally regarded as a software measurement method to the study of a non-software-specific problem.

Our study asked eight main questions regarding interrupts, which the following sections answer in detail.

What is the workload of interrupts?

Examining the effort spent on interrupts, we found that each requires approximately 20 minutes for occurrence and handling combined and that the average developer receives three to five interrupts per day. Thus developers spend roughly 1 to 1.5 hours per day on interrupts, which consume 15–20 percent of their total time. This is similar to the percentage found by Perry and colleagues.[6]

One project manager who participated in our research told us he always padded his planning estimates by 20 percent to account for unexpected activities such as organizational communication, training, socializing, and interrupts. However, our measurements showed him that this slack was already fully consumed by the interrupts occurring within his own department. Realizing that this might indicate a possible cause of planning delays, he acted to increase interrupt awareness and decrease the number of interrupts.

Through which medium do interrupts occur?

Figure 2 shows the distribution of interrupts across the three possible delivery media: personal visits, telephone calls, and e-mail. G1, G2, and G3 indicate the groups in which the measurements were performed. G1 and G2 are Dräger groups; G3 is the Schlumberger group.

In both organizations, personal visits and telephone calls caused 90 percent of all interrupts, and e-mail caused the rest. Significantly, interrupts resulting from personal visits or telephone calls automatically receive full attention, even though only 25 percent of all interrupts are classified as urgent. With face-to-face and telephone interrupts, the initiator determines the moment of interruption and the developer must respond immediately. However, the developer responds to an e-mail interrupt when it suits him or her. Further, personal visits and telephone calls need more handling than e-mail—apparently because e-mail interrupts are usually better formulated.

An Interrupt Example

Bob is a software developer primarily concerned with writing test applications. These test tools support a software package used to build graphical user interfaces for embedded payment systems. The tools that Bob develops help to automatically test a specific user interface by simulating possible user actions. This week Bob has been assigned to set up the design for a "record and playback" test tool to support regression tests. This tool will reuse several parts of a tool Bob developed last year. Bob has just accepted a cup of coffee from his colleague and returned to his investigation of components that could be reused in the new tool (Planned Activity).

The telephone rings (Interrupt Occurrence). Susan, one of the customers' developers, has just finished testing a new ATM user interface that makes it possible to charge banking accounts for fuel pumped at gas stations in Germany. Susan encountered some error codes when testing the user interface, but could not find these codes in the testing tool's manual.

Bob asks her for the specific error codes and explains to Susan which type of error probably is still in the product (Interrupt Handling). He promises to send her the latest version of the manual, because Susan apparently has an expired version. She asks for four copies because her colleagues are using the same outdated manual. She then thanks Bob and hangs up. Bob sends an e-mail to the marketing department requesting that four manuals be sent to Susan. He also reminds the department that they should send the new version to all other users as well, because he should not be disturbed to provide help with such issues (also Interrupt Handling).

Bob gets more coffee for himself and his colleague, then returns to his work (Planned Activities). It takes some time to regain his concentration (Recovery Time), but after a quarter of an hour he is working again at normal speed.

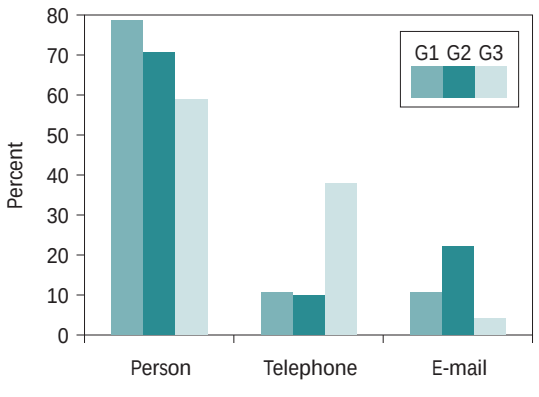

Figure 2. Interrupt occurrences classified by the three media in which they are delivered: personal visits, telephone calls, and e-mail.

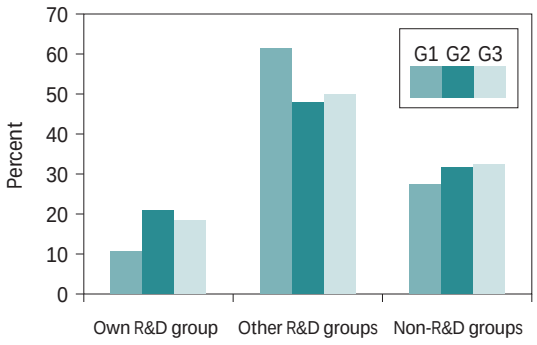

Figure 3. Interrupt initiators classified by category: members of the developer's own R&D group, members of other R&D groups, and members of non-R&D groups. The developers were surprised that less than a third of all interrupts are initiated by departments outside R&D.

Who causes interrupts?

Figure 3 shows the distribution of interrupts over the types of initiators: members of the developer's own R&D group, other R&D groups, and non-R&D groups.

All R&D groups combined accounted for fully 70 percent of the recorded interrupts. The developers found this remarkable, because they expected that other departments would be the main source of interrupts. This result suggests that interrupts are an R&D-inherent issue, and that solutions to this problem should be found primarily within those departments.

What is the recovery time after interrupts?

We initially included a metric for interrupt recovery time in our measurement programs. Unfortunately, this variable proved too difficult to measure, so after the first measurement period we omitted it from all programs. Nevertheless, we did observe that recovery time is primarily a problem when an interrupt occurs during actual programming work, and less so when it occurs during documenting or meetings, for example. We assume that the greater concentration required during programming accounts for the increase.

DeMarco reports that the recovery time after a phone call is at least 15 minutes.[2] Even though we could not measure recovery time exactly, we believe his estimate to be valid. If more than 10 interrupts occur during a day, the time between the interrupts becomes too short to accomplish product development work. This agrees with related work done by Perry and colleagues, who claim that developers typically perform their work in blocks of two hours.[6]

How are interrupts handled?

Figure 2 shows that fully 90 percent of interrupts are difficult for developers to avoid. However, some interrupts can be postponed, depending on type. Most interrupts are small questions or requests that developers prefer to handle immediately. Also, some interrupts cannot be postponed because the initiator considers them urgent. At Schlumberger we measured whether interrupts were handled immediately or postponed, and found that developers handled 90 percent of the interrupts immediately. However, the postponed interrupts needed three times more effort to handle. This may well be why they were postponed in the first place.

We suspect that lack of relevant details also caused developers to postpone interrupt handling. Both organizations assumed at the start of the measurement program that 50 percent of all interrupts have insufficient details. However, our measurements showed that 90 percent of all interrupts for both organizations were formulated with sufficient information. At Dräger, however, the 10 percent of interrupts that lacked sufficient detail needed three times more effort than at Schlumberger, where such interrupts took no longer to handle than others. We could not determine the cause of this disparity.

What are the underlying reasons for interrupts?

Figure 4 shows the leading causes of interrupts, including knowledge or experience exchange, social issues, documentation, organizational issues, and other causes.

For both organizations, approximately 25 percent of all interrupts were caused by issues related to knowledge or experience, such as discussing program structures or explaining technical details. Further, 20 percent were caused by documentation issues, such as documentation review and inadequate or nonexistent documentation. Both companies also suffered from organizational shortcomings in that 10 percent of all interrupts were caused by initiators who were unaware of the exact responsibilities held by the particular developer they approached.

What actions can be taken to solve the interrupt problem?

Developers in both organizations consider an interrupt to be inappropriate if it is delivered to the wrong person, or if the initiator himself could have handled it. Approximately 30 percent of all interrupts fell into this category, and an organization that wants to improve its interrupt efficiency should probably focus first on removing these. Given that unclear organizational responsibilities caused many interrupts, employees' individual responsibilities should be communicated throughout the organization.

To influence interrupt initiators, both organizations focused on establishing a company-wide awareness of interrupts' impact on productivity, actively communicating our measurement results via presentations, posters, and online resources. This interrupt awareness should contribute to initiators delivering—and developers handling—interrupts more efficiently.

Because developers handle e-mail interrupts more efficiently than those delivered by personal visits or telephone calls, both organizations set up and promoted an interrupt e-mail policy. The engineers agreed that urgent interrupts could be delivered by telephone, but not, preferably, by personal visits because these proved quite inefficient and often distracted others besides the developer.

What are the positive and negative aspects of interrupts?

Although many developers consider interrupts a necessary and thus positive part of their development work, they do consider them negative because sometimes concentration is disturbed. Also, as mentioned, many interrupts fall outside the developer's area of responsibility. Table 1 shows which aspects of interrupts developers perceive as either positive or negative.

Some interrupts were considered either positive or negative depending on context. For example, if an interrupt concerned part of a developer's task or supported problem solving, it was considered positive; if it was not part of the developer's task or did not prevent problems, it was considered negative. Developers consider the social element of interrupts very positive—primarily because software development is considered an interactive, social, and creative discipline highly dependent on good cooperation. Random interaction, therefore, appears to be necessary for development work, and supports creating an open culture even though it costs nonproductive time and might delay projects.

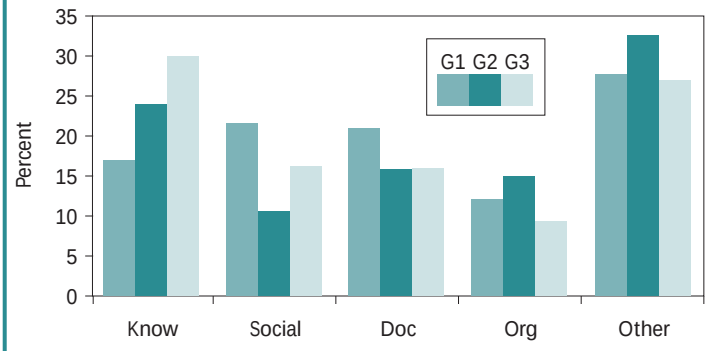

Figure 4. The exchange of knowledge or experience, social interaction, and documentation issues cause most interrupts.

Table 1
Interrupts' Positive and Negative Aspects

Positive	Negative
Part of developer's task	Disturbs developers' concentration
Supports problem solving	Requires additional interrupt effort
Stimulates social interaction	Not part of developer's task
Random communication essential for development work	Does not prevent problems
Provides overview of development work's current status	Causes project delay

Company-Specific Results

Our study answered the basic questions regarding interrupts by focusing on the similarities in both organizations' measurement programs. However, not all results were identical.

Dräger Medical Technology

At Dräger we measured the interrupts for two groups: hardware developers and test engineers. The hardware developers exhibited a negative attitude toward interrupts because such distractions disturb the main task, which requires high concentration. Test engineers regarded the interrupts as part of their job and useful.

Social visits and environmental noise caused 30 percent of all interrupts. Environmental noise was caused by, for example, conversing colleagues, operating machinery, and the cleaning staff. The developers ascribed the high percentage of such

distractions to their currently crowded work environment. A quieter workspace with more privacy should decrease the interrupt percentage significantly, so Dräger plans to rearrange its development environment along the lines DeMarco and Lister suggest in *Peopleware*.[2]

A quieter workspace with more privacy should decrease interrupts significantly.

Finally, Dräger's developers found especially unpleasant those interrupts caused by failure or insufficient knowledge of development tools. Even though few such interrupts occurred, the developers spent much effort handling them. Therefore, they suggested focusing special attention on keeping the tool environment running and providing adequate tool training.

Schlumberger Retail Petroleum Systems

At Schlumberger we measured interrupts within a team consisting of two hardware developers, three software developers, and their manager. While the developers considered interrupts a positive element in their work, the interrupt measurement program increased awareness of interrupts' negative effects, especially among the interrupt initiators. As a result, interrupts decreased by 30 percent during the measurement period alone.

We arrived at two practical conclusions from the Schlumberger interrupt measurements.

First, normal project plans included 20 percent slack for unexpected work. Yet it appears that this cushion is already spent on interrupts, which could explain planning delays. Obvious solutions are to further pad planning estimates or to reduce the number of interrupts.

Second, interrupts during meetings should be reduced to zero. Such interrupts prove very expensive because they disturb discussions involving many people.

Schlumberger achieved many improvements with regard to minimizing interrupts. Straightforward examples include reducing the number and duration of interrupts during meetings and a 50 percent effort reduction when dealing with those meeting interrupts that still occurred. The number of interrupts caused by documentation issues dropped by 80 percent thanks to improved documentation update and distribution practices.

Interrupts and Maturity Levels

Interrupts are necessary to solve development problems. However, it would be better if the problems causing the interrupts had been prevented by, for instance, improving software development according to maturity models such as the CMM.[9] Established processes prevent many problems, and result in proportionately fewer interrupts. For example, an improvement initiative that strives to improve requirements management will decrease many interrupts related to requirements changes or product nonconformance.

When we presented our first results on interrupt measurement at a well-established conference, an experienced assessor approached us afterward. He told us that he had visited a large telecommunications organization only the day before. This organization had started a company-wide improvement program to assess all software-producing units. While visiting one software department to plan his initial assessments, he noticed several groups of engineers gathered around computers, mainly solving urgent problems. The telephone rang constantly and engineers interrupted one another repeatedly. Yet this was not a help-desk department but rather the main software-producing unit.

The assessor told us he advised the department manager to postpone all initial assessments for at least three months. He recommended the postponement because the assessment he planned to carry out would mainly assess the "established process." When engineers have no time to perform that established process, the assessment is of little value. The assessor further recommended that the manager work on the interrupt problem first and, once he solved it, the assessor would return and perform his assessments.

At this same conference we spoke with a management consultant who told us that, according to his experience, our study's measurement—that 20 percent of project effort is devoted to interrupts—is too low. When we asked him for actual data, however, he admitted that the organizations he had in mind did not perform any data collection.

Because earlier research concluded that GQM measurement resembles organizational learning by development teams,[10] we expected that GQM is applicable to organizational problems such as in-

terrupt handling. Our research confirmed this.

The most important benefit of an interrupt measurement program is the creation of interrupt awareness. Both the interrupt initiator and the interrupted developer must become more aware of interrupts' negative effects. Although interrupts often result in a positive exchange of information that moves the project forward, *when* and *how* an interrupt occurs can determine the extent of its negative impact. Initiator awareness causes a more thorough assessment of the interrupt, which results in more coherent interrupts and combined interrupts. Developer awareness causes a more proper response to interrupts and the formulation of better plans for dealing with interrupt-related activities. This, in turn, results in increased interrupt-handling efficiency. Developers spend approximately one day a week handling interrupts; active management of interrupt generation and handling can reduce this time by more than 50 percent.

Both organizations performed their interrupt measurement programs for only three months. We realize that this gives us a limited data set, but our results look promising. We recommend further research into the organization of software development, because we expect that software research might overlook the benefits that can be achieved by improving non-software-specific processes. ❖

References

1. N.E. Fenton and S.L. Pfleeger, *Software Metrics, a Rigorous and Practical Approach*, Thomson Computer Press, London, 1996.
2. T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, Dorset House, New York, 1987.
3. R.L. Glass, *Software Creativity*, Prentice Hall, Upper Saddle River, N.J., 1995.
4. B. Curtis, W.E. Hefley, and S. Miller, *People Capability Maturity Model (P-CMM)*, CMU/SEI-95-MM-02, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1995, http://www.sei.cmu.edu/publications/documents/95.reports/95.mm.002.html.
5. W.S. Humphrey, *A Discipline for Software Engineering*, Addison Wesley Longman, Reading, Mass., 1995.
6. D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software*, July 1994, pp. 36-45.
7. V.R. Basili, C. Caldiera, and H.D. Rombach, "Goal/Question/Metric Paradigm," *Encyclopaedia of Software Engineering, Vol. 1*, J.J. Marciniak, ed., John Wiley and Sons, New York, 1994, pp. 528-532.
8. F. van Latum et al., "Adopting GQM-Based Measurement in an Industrial Environment," *IEEE Software*, Jan./Feb. 1998, pp. 78-86.
9. W.S. Humphrey, *Managing the Software Process*, Addison Wesley Longman, Reading, Mass., 1989.
10. R. van Solingen, E. Berghout, and E. Kooiman, "Assessing Feedback of Measurement Data: Practices at Schlumberger RPS with Reflection to Theory," *Proc. 4th Int'l Software Metrics Symposium*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 152-164.

Acknowledgments

We thank the Dräger Medical Technology (www.draeger.com) and Schlumberger Retail Petroleum Systems (www.slb.com/et/ms_petrol.html) project teams for their participation in the measurement programs. We also thank Hans Leliveld and Tom Dolan for their contributions.

About the Authors

Rini van Solingen is concurrently employed as a software quality engineer for Schlumberger RPS and by the Faculty Technology Management of Eindhoven University of Technology in the Netherlands. His research focuses on software process improvement for embedded systems development.

Van Solingen received an MSc in computer science from Delft University of Technology.

Egon Berghout is an assistant professor of information strategy and management of information systems at Delft University of Technology in the Netherlands, and associate of the M&I/Partners group of IT strategy consultants. His research focuses on problems with efficiency and effectiveness in information systems.

Berghout received a BSc in industrial engineering from Eindhoven Polytechnic, an MSc in information management from Tilburg University, and a PhD in informatics from Delft University of Technology. He is chairman of the Dutch Information Economics Working Group, program chair of the Fifth European Conference on the Evaluation of Information Technology, and member of the Information Resource Management Association.

Frank van Latum is R&D manager at Dräger Medical Technology. His main interests are the professional management of embedded systems development projects, with emphasis on product and process improvement.

Van Latum received an MSc in mathematics and an MSc in computer science from the University of Nijmegen. He is a member of the IEEE Computer Society.

Readers can contact the authors via van Solingen at r.v.solingen@tm.tue.nl.