Intelligent Assistance in German Software Development: A Survey

Jörg Rech, Eric Ras, and Björn Decker, *Fraunhofer IESE*

Researchers have studied and created a wide range of techniques to support software engineers during development. This article reports parts of the results of a survey of 135 participants that has been conducted in Germany to shed light on the usage and demand of intelligent assistance in software engineering activities. The survey showed that there is a high demand and acceptance for unobtrusive, quickly executable, and reactive assistance in core software engineering phases to help solve the problems at hand. In addition, several challenges for the future in software engineering work environments are pointed out.

While many environments are explicitly or implicitly using ideas from intelligent assistance research, the users are not always aware of its existence and potential. The goals of this survey are to clarify the concepts for intelligent assistance, describe the motivation for intelligent assistance systems, review some examples for intelligent assistance, and present the results of a survey about the attitude towards as well as the demand for intelligent assistance in German software organizations.

Assistance in SE

Intelligent assistance in software engineering is a relatively old research field (see sidebar) that is nevertheless of high interest for software engineers today. Giving support to the software engineers in programming, design, requirements, or other software-related environments is necessary, as the work product s typically very complex, large, and influenced by many persons.

The core objective of intelligent assistance is to enable and optimize the:

- *Automation* of simple or repetitive software development tasks such as compilation, test case generation, code smell discovery, etc.
- *Insight* into the system under development via cross-references, querying capabilities, or visualization.
- *Interaction* with and *negotiation* between the involved parties (e.g., the user(s) and / or the assisting (sub-)systems) to support cooperative work or to explain the assistance.

Intelligent Assistance: Past, Present, and Future

The roots of Intelligent Assistance in software engineering can be found in the early 1970s, when Terry Winograd wrote of intelligent assistance for programmers [1] and Teitelman described "The programmer's assistant" [2]. They set the stage for systems that should support developers with automated tasks – some that are ubiquitous today (e.g., undo/redo functionality) and some that are not (e.g., automated parameter checks). In the 1980s, environments such as Marvel [3] and "The Programmer's Apprentice" [4] for intelligent assistance were developed, which modeled the development process, automatically processed the software in the background (e.g., compile and analysis tasks), and gave additional assistance to the user if specific rules fired. The 1990s brought forth systems for requirements engineering such as "The Requirement's Apprentice" [5] or software design such as the "Design Apprentice" [6], "The Software Architect's Assistant" [7], or argoUML and its design critiques [8]. They assisted in tasks such as stepwise refinement, automated layout, consistency checking, or code generation.

Today, assistance is widely used in IDEs such as eclipse, IntelliJ, or Visual Studio .NET where information is offered regarding the compilation process (e.g., warnings and errors), the correctness of a function (e.g., verification of invariants in Spec# [9]), or by offering applicable refactorings. Other assistance features include syntax-highlighting, context assistance (e.g., context-sensitive help or code completion), wizards that generate running skeletons of applications, automated code inspections, or quick-fixes (i.e., small typical actions that would solve a current problem), to name just a few.

In the future, we might see even more intelligent assistance in environments for programming, maintenance, design, or requirements engineering that proactively support tasks such as software reuse, learning on demand, or automated product variant configuration, and bridge the gap between the phases of software development. Model-driven architecture (MDA) represents a kind of "programming assistance" that automatically translates models into source code – wrapping the required expertise for programming and about software (i.e., source code) qualities within. Furthermore, quality oriented assistance that reworks a system to emphasize a specific software quality such as maintainability or performance (cf. ISO 9126) could reduce the cost and time for software and system development.

References

1. Winograd, T. (1973). Breaking the complexity barrier again. Paper presented at the Proceedings of the ACM SIGPLAN SIGIR Interface meeting on programming languages - information retrieval, 4-6 Nov. 1973, Gaithersburg, MD, USA.
2. Teitelman W. (1972). "Automated programmering - the programmer's assistant," Proceedings of the Fall Joint Computer Conference, AFIPS, 1972, page 915-921.
3. Kaiser, G. E., Feiler, P. H., & Popovich, S. S. (1988). "Intelligent assistance for software development and maintenance." Software, IEEE, 5(3), 40-49.
4. Rich, C., & Waters, R. C. (1988). The Programmer's Apprentice: a research overview. Computer, USA * vol 21 (Nov. 1988), no. 11, p. 10-25
5. Reubenstein, H. B., & Waters, R. C. (1991). The Requirements Apprentice: automated assistance for requirements acquisition. IEEE Transactions on Software Engineering, USA * vol 17 (March 1991), no 3, p 226-240
6. Waters, R. C., & Yang, M. T. (1991). Toward a Design Apprentice: supporting reuse and evolution in software design. SIGSOFT Software Engineering Notes, USA * vol 16 (April 1991), no 2, p 33-44
7. Ng, K., Kramer, J., Magee, J., & Dulay, N. (1995). The Software Architect's Assistant-a visual environment for distributed programming. Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995.
8. Robbins J. E. (1999). Cognitive Support Features for Software Development Tools. Ph.D. Thesis. University of California, Irvine
9. Barnett, M., Rustan, K., Leino, M., & Schulte, W. (2005). The Spec programming system: an overview. International Workshop, CASSIS 2004. Marseille, France, 10-14 March 2004 * Berlin, Germany: Springer Verlag, 2005, p 49-69

Examples of Assistance

Today, a multitude of assistance systems are available that are more or less useful to a software engineer. In the following, several noteworthy systems are listed that are either broadly known or perceived as useful.

- One of the most widely known systems is probably the assistance in Microsoft's office programs (e.g., Clippy, Einstein, or Merlin) (see http://www.microsoft.com/msagent/). It assists users in working with the office tool and explains functions or error messages. While it is typically seen as not very helpful to experienced users, it might help novice users without a decent handbook.
- In the eclipse IDE (and many similar IDEs) many assistance systems were implemented, including auto-compilation, wizards that generate skeleton of common software systems (e.g., eclipse plugins), or features that automatically apply refactorings to a selected element. Furthermore, it offers code completion (i.e., it brings up a list of possible methods of a class, as well as the appropriate JavaDoc), the generation of small code fragments (e.g., getter/setter methods), and impact analysis on the code level (e.g., by showing problems if the signature of a method changes).
- Design critiquing systems such as argoUML include tightly integrated intelligent assistance that reports on design errors, incompleteness of the design, or interface mismatch. Furthermore, they are targeted to suggest alternative designs (e.g., due to problems in generating code from the design) or offer heuristic advice (e.g., a colleague who reported a problem).
- Another example for assistance is the Microsoft .NET language environment for "Spec#". This programming system includes the Boogie static program verifier. It assists the developer in formally specifying the source code and checks for the correctness in later change activities.

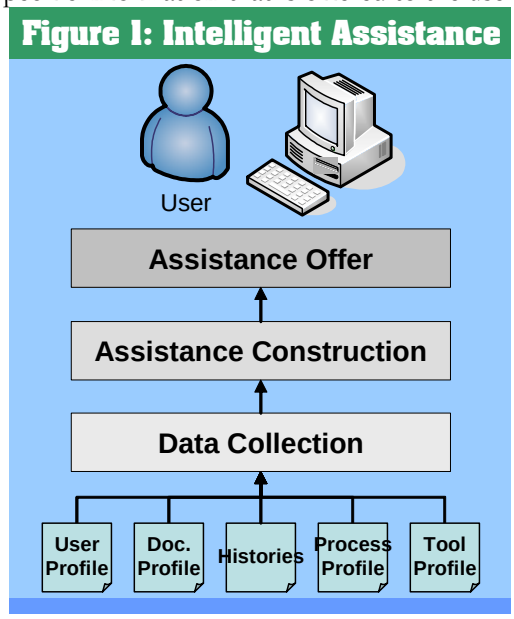
Of course, this is only a small subset of examples of existing assistance systems in contemporary tools used in software development. Today, almost every tool integrates assistance systems that are more or less "intelligent".

Dimensions of Assistance

While the range of possible assistance systems is very large, the groups of assistance systems can be distinguished by their information offering and data extraction characteristics.

The general data flow in an assistance system is depicted in Figure 1. Information from the content in work (the current document), the process (i.e., the current activity), the tool status, and information about the user is collected and made available in a unified, machine-readable format. This data is used by the core assistance algorithms to produce context-specific information that is offered to the user.

Figure 1: Intelligent Assistance



Offering Assistance

The assistance provided to the user is based upon the construction algorithm. While the result of this algorithm is fixed, the method of presentation can be differentiated by the following characteristics:

- *When to assist:* If every click by the user indicates a potential action, the question arises of when the user should be assisted. Assistance can be generated pro-actively (i.e., before an action), during actions, or after actions, and presented on demand or on request.
- *How to assist (media form)*: As modern computers often represent multi-media work environments, the form of media used by the assistance can be differentiated. Currently, assistance can be presented textually, visually (i.e., as a figure or animation), acoustically (e.g., an audio comment), or as a video.
- *Where to assist*: The information offered by the assistance system might be wrapped in tooltips, pop-ups, tables, specific sound effects (e.g., beeps), blinking effects, sidebars of a document, or specific marked spaces (e.g., views in the eclipse IDE). Furthermore, it can be presented within the active tool, a specific third-party tool, or in the operating system itself.
- *Why to assist*: What is the rationale for the assistance at all? There might exist a competence gap in the user's profile, a complex process step might be ahead, new tool features were integrated during an update, or a typically error-prone algorithm is currently being developed.

While an assistance tool has at least a fixed characteristic, it might also be possible that the system decides on its own what, when, and how the information should be presented to the user. This decision might be influenced by information about why and for whom assistance was generated.

Assistance Construction

Assistance comes in all sizes and flavors – from a simple tool explanation to an extended e-learning offering. While the functionality or result per se is hard to classify, the assistance algorithms can be characterized as follows:

- *Assistance for whom?* Who should be assisted with the constructed information? Depending on the user profile, the results need to be personalized or adapted to the expertise level.
- *Assistance about what*: What kind of object should be enriched with assisting information? Is it a requirements document, a testing tool, a process or activity model, general background knowledge, information about experts, etc.?
- *Assistance in which process?* The process or activity the user is currently involved in might induce a special need for assistance. For example, a programmer who is developing software (and is working on source code) has other requirements for assistance

than a tester or inspector who is looking at the same source code.

- *Assistance in which tool environment?* If the information about the process is not available, the tool environment context might be used as well in order to optimize the assistance. A running testing tool might be an indicator of the current process; using a text editor (e.g., OpenOffice) for a requirements document can imply missing knowledge or RE tools; and an updated coding environment (e.g., a new version of the eclipse IDE with several new plugins) might hint at new functions that are unknown to the user.

Data Collection for Assistance

Data from several sources has to be processed in the construction phase in order to generate information to assist the user. The characteristics that distinguish the approaches are:

- *Where to extract & preprocess data?* Several sources are available with information that might be useful to construct information for assistance. We can at least extract data from user, document, process, and tool descriptions that reside in the active tool (e.g., the document content) or external databases (e.g., LDAP server for user data).

Table 1

The size of the respondents organization

Organization Size	Employees	Percentage	Expected Percentage
Micro Organization (1-9 employees)	36	26,7 %	77 %
Small Organization (10-49 employees)	44	32,6 %	16 %
Medium Organization (50-250 employees)	24	17,8 %	5 %
Large Organization (250+ employees)	13	9,6 %	2 %
N/A	18	13,3 %	–

- *When to extract & preprocess data?* The optimal time for extraction depends on the type of assistance and variability of the data. A user profile will probably stay the same over the course of a few hours, while a document might change its content dramatically. The data can be extracted continuously during work, action-driven, or on demand. Furthermore, depending on the currently active assistance algorithms, the extraction might even be suspended.
- *How to extract & preprocess data?* The data from the different data sources has to be extracted and preprocessed. Extraction depends on where the data resides and the available interfaces or querying languages. Preprocessing, similar to the ETL process in data warehousing, offers several techniques for the unification or discretization of data to generate homogenous data for the next phase. Furthermore, there is the question of how does the system cope with missing, incomplete, inconsistent, or incorrect data? Some systems will ignore, correct, or filter the existing data while others will blindly use it.

The Survey

The survey about intelligent assistance in software engineering was conducted because of two reasons. First, it should elicit what practitioners demand from their SE environment and what kind of information they need and prefer. Second, it was targeted to capture the knowledge of practitioners about assistance as well as their opinion about the different kinds of assistance available today.

In order to elicit the current view of practitioners, the survey was conducted with German enterprises. A fairly recent study in the German software industry [1] determined the standard distribution of organizations in Germany as shown in the "expected percentage" column in Table 2. In comparison with this study we got far more answers from larger organizations (including Fortune Global 500 and multinational corporations) than from micro organizations. Our respondents consisted of a total of 460 individuals, of which 135 completely finalized the questionnaire – including 89 companies, 18 freelancers, and 9 research organizations. The study was conducted between 2 March and 9 April 2006.

The survey consisted of eight pages with a total of 38 questions (including additional explanatory information) that required an average of 30 minutes to answer. A short German summary [2] of the survey is available online at

http://www.iese.fraunhofer.de, which also includes additional questions and answers about used products and processes.

The questionnaire was designed using multiple choice questions (mostly based on a five-level Likert scale) wherever possible, as these are more likely to be answered, and it is easy to statistically analyze the answers. The two common types of multiple choice questions were "choose all that apply" and "choose the best". To allow unexpected answers, most questions had an "Other" choice with some extra space for one's own comments. It was discovered that this is especially useful when the range of answers might be too long (e.g., tools for requirements engineering) or if there is doubt whether the given answers are complete. This made it possible to elicit some previously unknown facts via the survey.

To develop the survey pages and make them available on the Internet, a commercial tool called OPST from the company Globalpark (http://www.globalpark.de/) was used. In order to collect a large set of participants, a commercial online company database by Hoppenstedt (http://www.hoppenstedt.de/) was used, which includes over 225,000 profiles of German companies, banks, their branches, and the major industrial associations in Germany. Furthermore, the group of software developers were addressed by subscribing to German mailing lists designated to software development and engineering activities. Bi-weekly reminders to the lists were used to inform the other subscribers about our survey.

Table 2

The number and percentages of the respondents assuming each role

Respondents Role	Number	Percentage
Executive Board	28	20,74 %
Middle Management	11	8,15 %
Department Heads	17	12,59 %
Project Manager	21	15,56 %
Employee	36	26,67 %
Other	5	3,70 %
No answer	17	12,59 %

Table 2 summarizes the respondent profile of our survey. Most respondents identified themselves as employees, followed by executive board members and project managers. The respondent profile obtained met our prior expectations, considering the basic user group of assistance in software engineering tools. Non-management employees and project managers are the group that is supposed to have the most contact with tools in this domain.

Findings

The following results are extracted from the answers to twelve questions from the explorative survey. We provide the main survey findings in graphical format for brevity. The following descriptive statistics are grouped into the three categories "Information Need", "Known Assistance Systems", and "Demand for Assistance". Other topics from our survey will be discussed elsewhere.

Information Need

The first question of the study was designed to elicit what type of information a software engineer needs that might be provided by an assistance system. The question stated in Chart 1 *"What kind of information is needed during work"*, provided several statements, and gave five options for answering them (i.e., from very often to never).

About 83% of the participants stated that they often / very often require reusable documents (e.g., source code, requirements, test cases, measurement plans, etc.) and 76% of the participants answered that they often / very often require templates or examples. However, information in the form of courses and tutorials are only required often / very often by 15% while 39% requires them rarely. This indicates that there is a high demand for information that is already available in companies or can relatively easily be extracted from available internal documents or the body of knowledge in software engineering (e.g., templates such as IEEE Standard 830).

Beside the required information, the survey should elicit in which phases the participants have the need for further information and, therefore, asked *"In which phase do you typically need additional information?"* Chart 2 shows that the main phases where additional information is needed are the core SE phases requirements elicitation, design, and programming as well as project management. In comparison, activities

such as measurement, versioning, and maintenance are not considered as phases where additional information is needed.

This indicates that there is a high demand for additional assistance systems in the core phases where many tools are available. If we compare the findings from other studies by Lubars et al. [3], Emam et al. [4], Nikula et al. [5], and Hofman & Lehner [6] this indicates that assistance is needed in phases were no or only very general tools such as office suites or web sites exists. The other phases that are either rarely used in SMEs (e.g., measurement) or are not very complex (e.g., versioning) do not represent fertile grounds for assistance. This is partially affirmed in the findings of the study by Koru & Tian [7] that found that "Developers and testers record defects fairly consistently and keep fairly complete defect records" and therefore assistance seems not required.

Since intelligent assistance should provide support for obtaining relevant information, the rationales of why information is searched and used (i.e., the retrieval rationales) give further input for determining the requirements for assistance. The retrieval rationales were determined using the question *"Why are you gathering information?"*. As depicted in Chart 3, the main rationales for gathering information are solving concrete problems, closing knowledge gaps, and personal motivation. Therefore, intelligent assistance should focus on the current demands and address current problems and knowledge gaps for the retrieval of information.

In order to provide the appropriate way of assistance, it needs to be determined which kind of information and which way of presenting information provides a high personal benefit for the user. Most demands implied by the retrieval rationales can be satisfied within a short period of time and focus on a concrete solution. This allows evaluating with reasonable effort whether a certain assistance was successful.

This short-term perspective on assistance concerning concrete topics can also be found in the retrieval rationales for learning (Chart 4) based on the question *"Which learning-specific aspects should be improved by assistance?"*. It shows that 82% of the participants agreed that solving a concrete problem is important. The other aspects were rated comparably low. Therefore, it would improve user acceptance if long-term competence development could be tightly integrated within a series of concrete problem solutions.

In summary, the following trends concerning information needs for intelligent assistance can be identified: a) solving concrete problems quickly is the area with the highest information needs, b) assistance is requested for core software engineering processes, and c) intelligent assistance can rely to some degree on the correctness of information existing within the organization.

Known Assistance Systems

This second section investigates which kinds of intelligent assistance are known by the participants and how they are used. These investigations can support the development of intelligent assistance in general by indicating which type and propagation of intelligent assistance is accepted by users.

Diagrams I

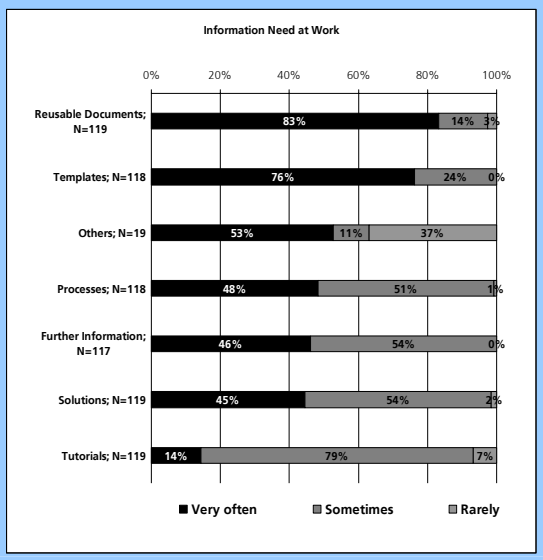


Chart 1: Required Information

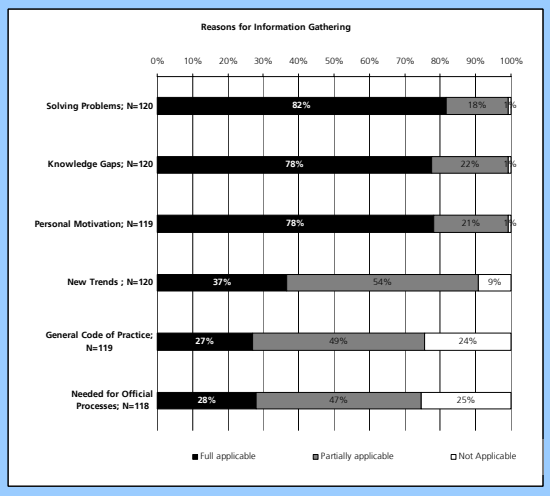


Chart 3: Retrieval Rationales

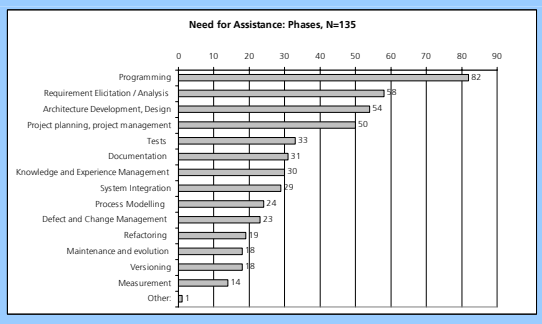


Chart 2: Phases Chart

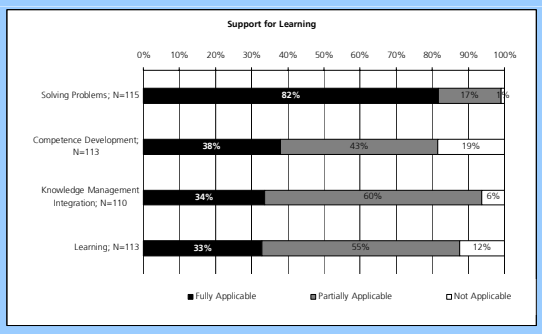


Chart 4: Retrieval Rationales 2

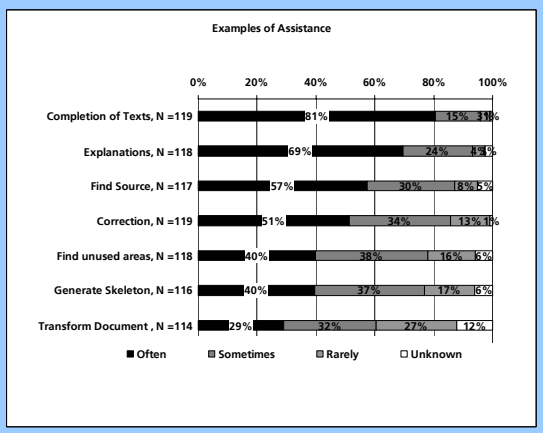


Chart 6: Known Examples

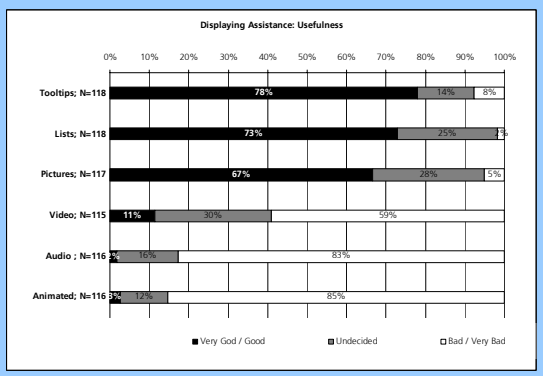


Chart 7: Forms of Assistance

As depicted in Chart 6, there is a tendency towards quickly executable assistance in a certain situation. An example of these kinds of assistance is the completion of texts. Complex kinds of assistance generating skeletons or transforming documents are used seldom or never and are relatively unknown. For these kinds of complex assistance, it needs to be determined how they are communicated to the user in an unobtrusive way. Furthermore, it needs to be determined whether these kinds of assistance are generally used less often because they generate results that are not needed so often during work.

The usefulness of the display format and media assistance is one major issue in designing intelligent assistance. As depicted in Chart 7, participants prefer simple, visually perceptible forms of intelligent assistance: tooltips, lists and pictures. Animated and audible forms of assistance are regarded as disturbing. One potential explanation is that these media types demand the full attention of the user and might distract from the current flow of work. This means that a sensible, textual support is sufficient to provide helpful assistance, which in turn lowers the effort to develop intelligent assistance.

This preference of unobtrusive forms of assistance can also be found in Chart 8. Reactive assistance, i.e., assistance that is explicitly requested by the user, is clearly preferred. However, one third of the participants prefer proactive assistance, which is displayed upon the decision of the assistance system. Therefore, it is an application-specific decision of whether to provide reactive or proactive assistance. In addition, if a proactive mode is available, an assistance system should also provide a reactive mode.

In summary, the analysis of known assistance system affirmed the trend towards simple and problem-oriented forms of assistance.

Demand for Assistance

This third section was used to clarify further requirements for intelligent assistance in Software Engineering.

Chart 10 gives an overview, of which kinds of assistance are wanted during the creation and editing of documents (multiple answers were allowed). Answering the question *"Which forms of assistance would you like for document creation or editing?"*, more than half of the participants stated that showing problems, completion of text, explanation of the currently edited docu-

Diagrams 2

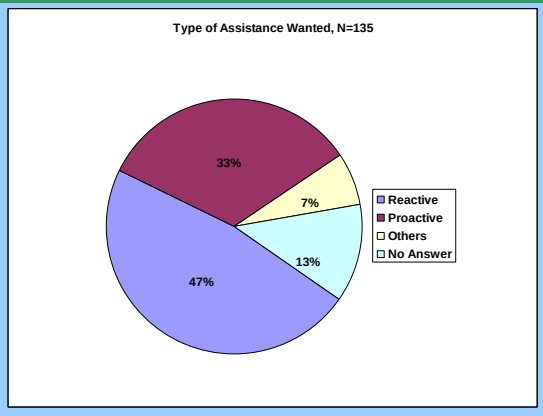


Chart 8: Type of Assistance

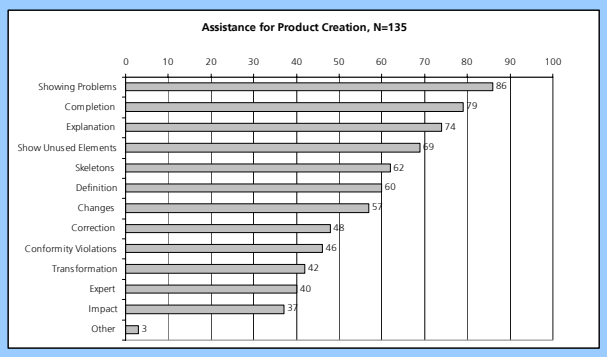


Chart 10: Content Need

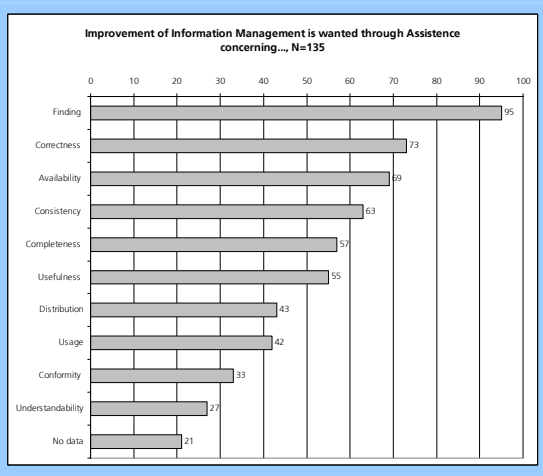


Chart 11: Information-Quality Need

ment, and highlighting unused parts would be desirable.

Chart 11 shows which aspect of information management should be improved by intelligent assistance and thus, gives directions for potential application areas. With multiple answers possible, more than 50% of the participants answered that finding information, correctness, availability (getting the found information) and consistency of the artifacts are potential areas of application for intelligent assistance. Furthermore, finding needed information is a predominant aspect that should be improved by intelligent assistance.

In summary, the participants state a general potential to improve software quality by intelligent assistance. Again, there is a tendency found towards quick solutions of concrete problems, with finding relevant information as the main application area for intelligent assistance.

Summary

The findings of this survey provide a general characterization of the information need and assistance in software engineering for an important subset of companies and can be used as a starting point for people interested in the development of intelligent assistance systems and related quality assurance and improvement activities. The survey results provided the following observations about intelligent assistance as perceived by German participants from SMEs (As this survey was conducted in Germany we can't generalize our findings globally. To test for regional bias the survey should be replicated in other countries):

- Intelligent assistance is mainly needed in core software engineering phases such as programming, design, or requirements development.
- There is a demand and acceptance for unobtrusive, quickly executable, and reactive assistance that help to solve the problems at hand.
- The phases where assistance is needed have sufficiently formal and available documents to generate plenty of supporting assistance systems.

The full survey report [2] contains further analysis about intelligent assistance in general (e.g., the time needed for information search) as well as an analysis of the tool infrastructure in which intelligent assistance has to be integrated (e.g., bug-tracking, versioning, requirements management systems). The findings are particularly interesting for companies developing tools for software engineers as well as managers responsible for the tool infrastructure and for employee training.

References

[1] M. Friedewald, H. D. Rombach, P. Stahl, M. Broy, S. Hartkopf, S. Kimpeler, K. Kohler, R. Wucher, and P. Zoche, "Status of the software development industry in Germany," *Informatik Spektrum,* vol. 24, pp. 81-90, April 2001 2001.

[2] J. Rech, B. Decker, and E. Ras, "Intelligente Assistenz in der Softwareentwicklung 2006: Zusammenfassung der Ergebnisse," Fraunhofer IESE, Kaiserslautern IESE Report: IESE-045.06/D, 2006.

[3] M. Lubars, C. Potts, and C. Richter, "A review of the state of the practice in requirements modeling," in *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, 1993, pp. 2-14.

[4] K. El Emam and N. H. Madhavji, "A field study of requirements engineering practices in information systems development," in *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, 1995, pp. 68-80.

[5] U. Nikula, J. Sajaniemi, and H. Kälviäinen, "A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises," Telecom Business Research Center Lappeenranta, Lappeenranta, Finland, Research Report 2000.

[6] H. F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *Software, IEEE,* vol. 18, pp. 58-66, 2001.

[7] A. G. Koru and J. Tian, "Defect handling in medium and large open source projects," *IEEE Software,* vol. 21, pp. 54-61, 2004.