User and Task Tailored Software Explanations

Ali Erdem, W. Lewis Johnson, Stacy Marsella
USC Information Sciences Institute & Computer Science Dept.
4676 Admarity Way, Marina del Rey, CA 90292-6695
+1 310 822 1511
{erdem,johnson,marsella}@isi.edu

Abstract
The main factors that affect software understanding are the complexity of the problem solved by the program, the program text, the user's mental ability and experience and the task being performed. This paper describes a planning approach solution to the software explanation problem that focuses on the user's task and expertise. This solution is implemented in a software explanation system as follows. First, user questions about software artifacts have been studied and the most commonly asked questions are identified. These questions are organized into a question model and procedures for answering them are developed. Then, the patterns in user questions while performing certain tasks have been studied and these patterns are used to build generic task models. The explanation system uses these task models in several ways. The task model, along with a user model, is used to generate explanations tailored to the user's task and expertise. In addition, the task model allows the system to provide explicit task support in its interface.

Keywords
software explanation, software understanding, user model, task model, explanation planning

1 Introduction and Motivation
Software maintenance has become an important activity in the software industry. Maintenance of existing systems consumes 50% to 75% of the total programming effort [17] and a significant portion of this maintenance activity (30% to 60%) is spent on software understanding [5, 3].

Software Understanding is the reconstruction of logic, structure and goals of a program in order to understand what the program does and how it does it [2, 3]. This reconstruction process is typically composed of inquiry episodes [3, 16] which involve the following steps: read some code, ask a question about the code, form an hypothesis and search the documentation and the code to confirm the hypothesis. The generation and verification of the hypothesis are influenced by the salient features in the code and the documentation [3].

Attempts to solve the software understanding problem have focused on two methods: improving the search and automating the recognition of features in the code. The search process was improved either by providing an automated search capability or by changing the organization of the documents. For example, Devanbu's LaSIE system [6] used description logic to represent the domain and the basic software knowledge. Users could do searches by forming queries using the predefined domain and software concepts.

Linked and layered documentation organizations have also been used to improve the search process. Soloway linked parts of the documentation to delocalize the programming plans [16]. SODOS project at USC [20] linked all the Software Life Cycle documents of a software project and also provided search capabilities. Rajlich [13] organized documentation into a problem domain layer, an algorithm layer and a representation layer. Users easily guessed which layer would have the answer to their queries and restricted their search to that layer.

The Programmer's Apprentice [14], Hartman's UNPROG [8] and Will's GRASPR [21] programs tried to automate the recognition of the standard programming plans in the code. This approach improved both recognition of the features in the code and reduced the search space from the actual code to the programming plans.

These improved search and automated recognition methods ignored one important factor, the user. Software understanding is affected by the complexity of the problem solved by the program, the program text, the user's mental ability and experience, and the task being performed [3]. The methods described above focused on the first two factors, but ignored the latter two. Even with good documentation, users still prefer asking questions to system experts or other users before consulting the documentation. There are several reasons for this behavior. First, the dialogue between the user and the expert facilitates the refinement of the dialogue permits the users to ask follow-on questions and clarify the parts they did not understand. In addition, the human experts can recognize the users' plans and provide answers to satisfy their goals [4]. Finally, the experts also recognize the users' level of expertise and provide tailored answers that are easier to understand.
Our research goal is to develop an interactive software explanation tool that can act as a system expert and provide tailored explanations to user questions that are easy to understand and are relevant to the user goals. Such a tool needs to have the necessary knowledge to answer the user questions. To identify what type of questions are asked by the users and what governs the answers the experts provide, we studied the questions posted to the USENET Tcl/Tk newsgroup [9]. Based on this study and a survey of the research literature on questions, we later developed a domain independent question model for software understanding questions.

The results of the USENET study have been used in the implementation of an initial version of an online explanation tool called MediaDoc. MediaDoc's knowledge representation is organized around the question model. Further, a planning component has been added to MediaDoc for user tailoring. The planner uses the question model along with a user model to determine the explanation content. Procedures for answering the individual questions in the model are built and these procedures are used in generating the explanation from the explanation plan.

This initial version of MediaDoc generated tailored answers to user questions. The tailoring was done using the user supplied expertise and role parameters. Although this version was capable of generating tailored answers to individual questions, it had two important shortcomings. First, there was insufficient task support. In the USENET study we observed questions like What do I do now?, How can I do task? etc. that asked about how to achieve a goal. However, the question model focused only on the questions about the software artifacts and did not include these task related questions. As a result there was limited support for answering these types of questions. Second, since the users' immediate goals were not known this version was unable to tailor the explanations to the users' goals.

Researchers have shown the importance of taking user's goals and plans into account in information seeking dialogues [10]. We observed the importance of user's plans and goals, both in the questions and the answers, in the USENET study. The most frequently asked questions included descriptions of what the users' goals were. If the users had attempted to solve the problems themselves, the messages also included their plans. In addition, the experts' tailored their answers by including only the information relevant to the user's goals and plans.

Users in the USENET study described their goals in detail because there was no surrounding task context. This placed an overhead on the interaction. Clearly, an automated explanation tool that requires the users to specify their goals in detail at every step would be unusable. To address this problem we needed a better understanding of the dynamics of the inquiry episodes and their relationship to the user's task. However, the USENET messages typically contained only a single inquiry episode and were not very useful for such a study.

To determine the dynamics of the inquiry episodes and to investigate the types of questions users ask within the context of a task, we decided to study users while they performed a task. We were particularly interested in finding out how the users started their task, the goals they tried to achieve at each step and the relation between the user goals and the questions they asked.

We studied two users while they performed different tasks on two different large software systems and recorded the questions they asked. Although the program domains were different, there was significant overlap in terms of the types of questions asked and the goals of the task steps. We observed distinct user goals that affect the types of questions asked. Using these goals and the patterns in user questions, we developed generic task models. These generic task models serve two purposes. First, they are used for giving explicit task support and answering the user questions about the task. Second, since the immediate user goal can now be identified by the task model, MediaDoc can generate explanations tailored to the user goal.

In this paper, we will describe the question model briefly to show the types of questions answered by MediaDoc. The focus of the paper however will be on the task study. We will present the details of the study and the observations we made. Finally, we will give examples that show the user and task tailoring in MediaDoc.

## 2 Question Model

Questions are the basis of user's interaction with the documentation and the system experts. Wright claimed that people's interaction with documentation starts with formulating a question, therefore the documentation content needs to be determined by the questions users ask [22].

We studied the questions asked by Tcl/Tk programmers in a USENET newsgroup and identified the most commonly asked question types [9]. It was possible to request the same information in many different ways in natural language. For our purposes, What does X do?, What is the function of X?, What does X cause? all requested the same type of information. We classified the questions in the data set based on the type of information requested. In addition, since the USENET data set was biased and did not include questions for all software engineering tasks [9], we decided to survey the research literature for other studies on questions. After reviewing Lehnert's [11] and Graesser's [7] question answering systems, Swartout's research on questions asked during expert system explanations [15], Letovsky's research on questions asked during inquiry episodes [12] and Serbanati's list of most commonly asked types of information by programmers [15], we developed a question model. In this model, a question is represented based on its topic, question type and the relation type.

- **Topic**: The question topic is the entity referenced in the question. It can easily be identified as the subject of the question. For example, in What does procedure open do?, the topic is procedure open.
Table 1: Simple Questions

<table>
<thead>
<tr>
<th>Verification</th>
<th>Identification</th>
<th>Procedural</th>
<th>Motivation</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is</td>
<td>What</td>
<td>How</td>
<td>Why</td>
<td>When</td>
<td>Where</td>
</tr>
<tr>
<td>(Is)</td>
<td>(What)</td>
<td>(How)</td>
<td>(Why)</td>
<td>(When)</td>
<td>(Where)</td>
</tr>
<tr>
<td>Does it exist?</td>
<td>What is it?</td>
<td>How does it work?</td>
<td>Why is it necessary?</td>
<td>When does it exist?</td>
<td>Where does it exist?</td>
</tr>
<tr>
<td>Does it have inputs?</td>
<td>What are its inputs?</td>
<td>How are the inputs processed?</td>
<td>Why are the inputs necessary?</td>
<td>When are the inputs provided?</td>
<td>Where are the inputs?</td>
</tr>
<tr>
<td>Does it have outputs?</td>
<td>What are its outputs?</td>
<td>How are the outputs processed?</td>
<td>Why are the outputs necessary?</td>
<td>When is the output received?</td>
<td>Where are the outputs?</td>
</tr>
<tr>
<td>Does it have structure?</td>
<td>What is its structure?</td>
<td>How is it structured?</td>
<td>Why is it necessary?</td>
<td>When is it structured?</td>
<td>Where is it structured?</td>
</tr>
<tr>
<td>Does it do anything?</td>
<td>What does it do?</td>
<td>How does it work?</td>
<td>Why is it used?</td>
<td>When does it work?</td>
<td>Where is it work?</td>
</tr>
<tr>
<td>Is it used?</td>
<td>What is it used?</td>
<td>How is it used?</td>
<td>How does it arise?</td>
<td>When does the goal arise?</td>
<td>Where does the goal arise?</td>
</tr>
<tr>
<td>Does it have a goal?</td>
<td>What is its goal?</td>
<td>How does the goal arise?</td>
<td>Why are the conditions necessary?</td>
<td>When do the requirements arise?</td>
<td>Where are the requirements?</td>
</tr>
<tr>
<td>Does it have requirements?</td>
<td>What does it require?</td>
<td>How can the requirements be met?</td>
<td>Why are the satisfied conditions necessary?</td>
<td>When does the post conditions arise?</td>
<td>Where are the post conditions?</td>
</tr>
<tr>
<td>Does it satisfy?</td>
<td>What does it satisfy?</td>
<td>How are the conditions satisfied?</td>
<td>Why is the context necessary?</td>
<td>When is the context ready?</td>
<td>Where is the context?</td>
</tr>
<tr>
<td>Does it have a context?</td>
<td>What is its context?</td>
<td>How does it interact with its context?</td>
<td>Why is the context necessary?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Question type**: The question type identifies the type of information requested. It is one of verification (is), identification (what), procedural (how), motivation (why), time (when) or location (where).

- **Relation type**: The relation type identifies what kind of information about the question topic is requested. We identified the most commonly asked relation types from our data set, but further additions to this set are possible. The identified relation types are as follows:

  - **Topic (self)**: These questions ask about the question topic itself, e.g. *What is an integer? How does function X work?*
  - **Behavior (input/output)**: These questions ask about the input/output relationships, e.g. *What are the inputs to X?*
  - **Structure**: These questions ask about the structure of the topic, e.g. *What are the components of Y?*
  - **Function (cause)**: This type of questions ask about the causal relations between topics, e.g. *What does X do?*
  - **Use**: This type of questions ask about the usage of the topic, e.g. *What uses X?*
  - **Goal/Purpose**: These questions ask about the goal of the topic, e.g. *What is the goal of X?*

  - Whereas the other asks for the reason of existence.

- **Require**: These questions ask about the requirements (preconditions) of the topic, e.g. *What needs to be done before function X is called?*

- **Satisfy**: These questions ask the post conditions the topic satisfies, e.g. *What is true after function X is called?*

- **Context**: These questions ask about the relationship between the topic and its environment, e.g. *What systems does X windows run?*

More question types and relation types can be added to this model if necessary. For example, a **Who** question type can be added to answer questions about the ownership of topics. Some question type, relation type pairs do not apply to all topics, e.g. data items do not have processes and it is not very meaningful to ask how an integer variable works.

We categorized the questions into two groups based on how the answers can be calculated:

- **Simple questions**: These questions can be answered by simple data retrieval and are shown in table 1. Simple questions are represented as a three tuple (topic, question type, relation type), e.g. *How does procedure open work?* is represented as (procedure open, how, self), *What are the parts of student record?* is represented as (student record, what, structure).
Task Model

Users frequently described their plans and what they were trying to achieve in the messages posted to USENET Tcl/Tk newsgroup. The answers provided by the experts were also tailored to the users' goals. This showed the importance of producing explanations that are tailored to the users' goals and plans. However, it is unreasonable to expect the users to specify their goals at every step when they consult a human common human problem solving knowledge.

Identifying the user task is one way of finding out the users go through many inquiry episodes while they perform a task. We were particularly interested in finding the answers to the following questions:

- Where do the users start their task? How do they identify the system components relevant to the task at hand?
- Is there a structure to the task being performed? Do the users move from one inquiry episode to another based on some criteria? Do they try to satisfy particular goals at each step?
- What type of goals do the users try to achieve at each step? What type of questions do they ask? Is there a relation between the type of the questions and the goals?
- Can all user questions be represented by our question model?

To find the answers to these questions, we studied two users performing different tasks on two large software systems. In the first study, the task was to make a modification to a large software system and analyze the impact to other system components. The user performed the task in a day and recorded the individual steps he took along with the reasons for taking them. In the second study, the user was given a report of a problem that happened during system integration and was assigned the task of finding the source of the problem. In this study, the user did not perform the task, but rather described the interviewer the steps he would take and the types of questions he would ask at each step.

3.1 ModSAF Impact Analysis Study

ModSAF (Modular Semi-Autonomous Forces) is part of a distributed simulation (DIS) system for training that creates a virtual environment where trainees in simulators can interact with virtual automated forces which ModSAF simulates. ModSAF is written mainly in C and the version we were employing (version 2.1) had over 250 libraries and over 1.5 million lines of code.
The task to perform was to study the impact of creating an interface to the aircraft simulation components that would allow the dynamic modification to the route flying behavior. The subject's knowledge of ModSAF was based on five months experience in ModSAF, modifying it to build an interface that reported back the status of automated ground vehicles. Included in that knowledge was the awareness that ModSAF's simulations of an automated entity performing some mission were built from collections of lower level behaviors called tasks. The resources available for this impact analysis included the source code and the documentation which was in GNU info file format. In addition, the subject had available, and knew how to use, a specially instrumented version of ModSAF that could report what low level tasks were being invoked when a vehicle was performing some mission.

The subject took the following steps to complete the task:

- **Step 1. Comprehend the desired behavior:** The first step the subject took was to understand the desired modification. He was already familiar with ModSAF and knew what was meant by the dynamic route flying behavior.

  The questions he asked in this step were *What is meant by* dynamic modification to the route flying behavior? *What is the desired behavior of the modified system?*

- **Step 2. Identify the relevant components:** The second step the subject took was to identify the relevant system components. The subject ran a special instrumented version of ModSAF, which told him what ModSAF tasks had started or stopped, in order to identify the libraries that were involved in route flying behavior. The subject ran two different missions, a sweep mission and an attack ground target mission to identify the libraries. Running two missions was necessary for identifying if there was more than one implementation of vehicle navigation in ModSAF.

  The question he asked in this step was *What libraries are used in implementing route flying behavior?*

- **Step 3. Test whether route modification will work:** In this step, the subject modified the routes using ModSAF's task modification and GUI interface while the missions were running. His goals were to identify whether it was possible to modify the routes at all and also to find out the libraries involved in the route modification. Route modification with task modification interface was successful, but changing the route through the GUI interface failed.

  The question he asked in this step was *Is it possible to change the flight routes during a mission?*

- **Step 4. Comprehend unknown concepts:** The subject found out the names of the libraries involved in the previous steps and using ModSAF's naming conventions, he was able to find the documentation for these libraries. He read the documentation to understand what these libraries were and how they worked.

  The questions he asked in this step were *What is library X? What does it do? How does it work?*

- **Step 5. Comprehend how the system currently works:** In this step, the subject looked at the implementations of the libraries he thought were relevant to the route flying behavior. He looked at the route data structure, the parameters of the flight task data structure and the finite state machine code for the route flying task. He investigated why route modification through the task modification interface was successful and why GUI interface modification failed. He concluded that routes were assigned persistent object identifiers and in GUI modification case this identifier becomes invalid resulting in termination of route flying. He also investigated how groups of vehicles follow a route.

  The questions he asked in this step were *How is route implemented? What is the data structure for a route? What are the parameters to the task data structures? What are the dependencies between libraries?*

- **Step 6. Find methods for implementing the desired behavior:** Based on his observations, the subject came up with a list of suggestions for implementing the desired behavior. He did not actually implement the suggested modifications. He used his general programming knowledge to come up with the suggestions. He did not record any questions in this step, but we would expect him to ask some questions, especially when there are alternatives for implementation.

### 3.2 Integration Task Study

The system used in the second study was an aviation software. The engineer who performed the task was an expert in this area. He was given a post integration problem report on an airplane and was asked to find the source of the problem. The problem was a corrupted display service message during initialization. It only occurred once during the initialization stage and was a transient problem. The effect was either the loss of radar video or connection of the display unit to the wrong radar buffer.

The information sources that were accessible by the engineer were the video recording of the problem, the message bus traffic log from the airplane, a message database which contained the descriptions of all message types, the requirements documentation, the program code and the system experts, i.e. programmers and requirements people involved. The subject was not familiar with the details of the radar and display unit components, but he was familiar with the overall system architecture and had extensive knowledge in troubleshooting these kinds of problems.
The subject did not actually perform the task, but described the interview the steps he would take and the questions he would ask at each step.

- **Step 1. Identify the problem:** The first step was to understand exactly what the problem was. The engineer read the problem description carefully and identified all the terms he did not know. He would talk to the experts who wrote the problem description and the programmers to clarify the unknown or ambiguous terms.

Some of the questions he asked in this step were **What is the term? What does this term mean?** What is the relationship between the term and the problem?

- **Step 2. Comprehend the desired behavior:** In the second step, the subject wanted to know the desired behavior. He would consult with the experts and the requirements documentation to find the answer. He would first attempt to do a keyword search in the requirements documentation. If the keyword search failed to produce an answer, he would do a sequential search on the table of contents. His search would also attempt to identify the messages related to the correct behavior from the requirements documents. The search process was an iterative one. After reading a requirements document found with a keyword search, the subject would add new keywords to his list. The process would continue until all the relevant requirements were satisfactorily covered.

The questions asked in this step were of the form **What are the requirements that include these keywords? Is the requirement relevant to the problem? What does the requirement say about the desired behavior?**

- **Step 3. Identify the messages used in implementing the requirements:** At this point, the subject was trying to understand the desired behavior. He assumed the problem was due to incorrect implementation of requirements, since if it were due to incorrect requirements he would have noticed it in the previous step. So he started the search to identify the messages used in implementing the behavior. His main information source in this step was the message database. He would do a keyword driven search in the message database to find all the messages that were involved with the behavior. The questions he asked in this step were in the form **What message descriptions include this keyword? Is this message relevant to the problem? What message descriptions include this keyword? Is this message relevant to the problem?**

- **Step 4. Test the hypotheses generated in previous steps:** In this step, the subject tested the hypotheses generated in previous steps and asked whether he found the source of the problem. He would do a keyword driven search in the message database to find all messages that were involved with the behavior. The questions he asked in this step were **Where do users start their task? What are the requirements that include these keywords? Is the requirement relevant to the problem? What does the requirement say about the desired behavior?**

- **Step 5. Identify the problems in the bus traffic:** If the problem had not been solved yet, the subject would generate new hypotheses by looking at the bus traffic. He had some heuristics to identify the problems. For example, he would check to see the correct ordering of messages. If the keyword search failed to produce the answer, he would do a sequential search on the table of contents. His search would also attempt to identify the messages related to the correct behavior from the requirements documents. The search process was an iterative one. After reading a requirements document found with a keyword search, the subject would add new keywords to his list. The process would continue until all the relevant requirements were satisfactorily covered.

The main question in this step was **What could be wrong?**

3.3 Observations

- There are similarities between the two tasks. Although the tasks and the problems were different, there were similar steps in the problem solving behavior. These were not necessarily executed in the same order, but both subjects had a gather step in which they identified the relevant concepts, a comprehend step in which they read the documentation to understand these concepts, a test step in which they tested the hypotheses. The second subject was required only to find the source of the problem. His task did not include making modifications. That's why he did not have a final method step like the first subject.

- Where do users start their task? Both subjects had an initial information gathering step to identify the relevant topics. The first subject ran an instrumented version of ModSAF to identify which tasks started and stopped. The second subject preferred a keyword search on requirements to start. There are many ways of satisfying information gathering goal and some of these are domain dependent. For example, an instrumented version of the program might not be available to all the users. However, some techniques like keyword searches are domain independent and would be useful in all domains. The method used for information gathering depends both on personal preferences and the costs associated. For example, in the integration task study running the instrumented avionics software in the lab was
costlier than doing keyword searches on the requirements and was not preferred.

- **Was there a structure to the task being performed?** There were similar steps used by both subjects and the problem solving behavior seemed to be structured. The ordering of the steps, however, was not exactly the same and the subjects did not execute the task steps sequentially.

- **Was there a relation between the goal and the type of question?** The subjects asked similar questions during the task steps. Gather task was addressed by the search process, comprehend asked what is (term)? and how does (term) work? types of questions and test asked verification questions.

- **Sufficiency of the question model:** Since the task studies were performed offline, we didn’t have a list of actual questions that would be asked while the users performed the task, but rather a list of the types of the questions they asked. As a result, we tested the sufficiency of the question model based only on the type of questions. Out of 16 of the 22 types of questions observed in the task study were supported by the question model. This ratio would have been higher if we had looked at the actual questions, because some types of questions like What is term? How does term work? would be asked more frequently than others.

Some question types were not supported by the question model. The main problems we observed with the question model were as follows:

- **Weak support for gather task:** The question model requires a well defined question topic and does not support gather type of questions. These questions are intended mainly for search and usually include a description of the search criteria. However, they are search mechanisms which do not use the question model in MediasDoc, and these types of questions are supported by them.

- **No support for what could be wrong?** These questions require the system to have problem diagnostics skills and are not well supported.

- **Hierarchical and multi-layered mental representations:** The programming process constructs mappings from the problem domain to the programming domain, possibly through several intermediate domains and software understanding is the reconstruction of these mappings [3]. As a result, the experts’ mental representation of computer programs are hierarchical and multi-layered [19]. In our studies, we observed confirming evidence for this hypothesis. The first subject constructed mappings from the vehicle behaviors of the running program to ModSAF tasks from ModSAF tasks to finite state machines and finally from the finite state machines to the code. The second subject constructed mappings from the requirements to messages, and from these messages to the code. Both subjects’ hypothesis generation was top down and driven by these layers.

- **Heuristics for ordering hypotheses:** Both subjects used some criteria for ordering and testing hypotheses. The first subject took first at the route behavior of individual units then groups. His ordering criteria was based on the complexity of the behavior. The second subject explained the criteria he used as “How likely it is for this to be the problem and how easy it is to test it”. He assumed that the problem wouldn’t be trivial, since the programmers usually did not make trivial errors and the components were already unit tested before the integration.

- **Users desire automated support for mundane tasks:** Both subjects claimed that their performance would have improved if they had a tool to support their gather, identify tasks. They both had to shuffle through the documentation and do keyword searches to find out the information they were searching.

3.4 Software Engineering Tasks

We are using the results of the task study for building a task taxonomy. In this taxonomy, each task is associated with a specific user goal. Since, studying two tasks is not sufficient for identifying all possible task steps and the user goals addressed by these, we investigated the software engineering task descriptions used by other researchers. The taxonomy in figure 1 is prepared after reviewing the task definitions in...
Analysis
identification
comprehension
classification
assessment
Synthesis
design
planning
modeling
Fault finding
diagnosis
repair
verification
Configuration
Specification
Documentation
Explanatio
Management

Figure 1: A taxonomy of software engineering tasks

Serbanati [15], CommonKADS [1] and Programmer's Apprentice [14]. In this taxonomy, gather is an identification task, comprehend is a comprehension task and assess is an assessment task.

The types of questions asked by the users while performing the tasks in this taxonomy will be different. For example, the questions asked during the design task might focus on design alternatives and their impact whereas the diagnosis task might focus on fault models. Although the question model was developed after studying the software understanding questions and thus is biased towards analysis, it could easily be extended to cover the other types of questions. This way, the task support in MediaDoc can be generalized from software understanding task to other software engineering tasks.

However in this case the answering of the user question might need to be extended beyond explanation generation. The question model is useful for identifying the information needs of the users, but currently MediaDoc uses this information only for generating textual and visual explanations. Each task in the taxonomy might require different type of answer generation. If the information needs of the user can only be satisfied by invoking a debugger during a repair task, MediaDoc can easily be extended to invoke the debugger instead of generating an explanation. In this respect, MediaDoc can also be used as an environment for integrating software engineering tools.

4 Requirements for MediaDoc

After the task study, we wanted to incorporate the results of the study into MediaDoc. The requirements for this modification were as follows:

- **Provide explicit support for the user task**: Explicit task support is critical for novices and useful for experts. It is necessary to show the users what they have done, what they are currently doing and what else they need to do.

- **Support inquiry episodes using the question model**: The software understanding process starts with either no knowledge of the system or some knowledge gained in the past that may or may not be correct. Then the users generate and test many hypotheses during software understanding. At any point until the task is completed, the users will have a partial understanding of the system and a list of hypotheses they want to test. MediaDoc should help the users by keeping track of their beliefs about the system and the hypotheses they want to test. It is impossible to know what beliefs the users construct after reading the explanations, so instead of keeping track of users' beliefs, MediaDoc should keep track of the questions asked. Similarly, the users should be able to record the questions they want to ask in later stages. This way, the users can focus on which question to investigate next rather than trying to remember all the questions they have.

- **Tailor the explanation to the user goals**: The users will have distinct goals during different task steps. MediaDoc should take the users' immediate goals into account in explanation generation.

- **Provide support for the gather task**: Users start their task by an information gathering step. Support for this step needs to be improved in MediaDoc. Techniques like keywords searches and query mechanisms can be used to support the gather task.

- **Provide support for hierarchical and multi-layered representations**: Users construct mappings between layers of abstraction during the understanding process. MediaDoc should keep track of these mappings and help users recall the mappings they used.

These requirements are realized by adding a task agenda component to MediaDoc. Generic task models that capture the user goals and task steps independent of the question topics are developed. These generic models also capture the layers of abstractions present in the system. The users instantiate these generic models when they start a task. Then the user can interact with the task agenda by posting topics of interest and questions to the agenda. At any point only one of the task steps in the agenda is active and the user goal associated with that task step is assumed to be the current user goal. MediaDoc uses this current goal for tailoring the explanations to the users' goals.

The next sections will describe the MediaDoc architecture and give examples of user and task tailoring.
The current MediaDoc architecture is given in figure 2. The architecture is divided into two sections. The components in the upper part are used for acquiring and representing the knowledge necessary to generate the explanations. This knowledge is acquired from many different sources and stored in a central database that is organized by the topic name and type. Figure 3 shows some of the entries in the ModSAF domain database. The lines starting with A contain information about topics. This information consists of a unique identification number, topic type and topic name. The lines starting with an L contain information about the relations between topics. This information consists of identification numbers of the related topics, a relation type and a relation name.

The topics can be entered to the database directly by the domain experts, extracted from the source code by code analysis tools or retrieved from the documentation and other information sources by text extraction tools. The explanation repository is a text database of topic annotations that are provided by the domain experts. These annotations are used in answering the questions about the topics and can be text segments, canned text or general procedures for generating the explanations. We have also built knowledge acquisition tools for extracting knowledge about topics and answers to the questions about these topics from natural language documentation. The question model is used as an upper model in these text extraction tools.

The components in the lower part of the architecture diagram are used in the explanation generation and will be described in the next section.

6 User Tailoring

User tailoring is done by the explanation generation component. When a user asks a question, this question is communicated by the web interface to the planner. The planner then creates a top level goal for explaining the question to the user. This top level goal is represented as (knows user topic question-type). For example, when the user guest asks how function-X works?, the top level goal will be (knows guest function-X how self).

Then, the planner tries to create an explanation plan using the plan operators. The plan operators interact with the user model and the domain model to create a user tailored explanation plan. The user model contains information about the user's domain knowledge, his preferences, his task and level of experience. For example, if the user guest is a novice programmer and knows what function-X is, the user model will contain the facts (expertise guest novice), (task guest programmer) and (knows guest function-X what self).

A sample plan operator is shown in figure 4. This plan operator can be summarized as if a user asks how something works and if he prefers detailed explanations then explain what that thing is first if he doesn't already know. So in our example above, if the user guest did not know what function-X was then the system would have explained what the function-X is before explaining how it works. After an explanation, the user model is updated to reflect the fact that user guest now knows what function-X works. Note that in the user guest example above, user guest knew what function-X was before the query. As a result, the pre-condition of this plan operator would have been satisfied and the system would not have explained what the user already knew.

Some of the tailoring methods implemented by the current plan operators are as follows:

- Don't tell the user what he already knows
- Tailor the explanation to the user's role, e.g. if an end user asks what something is, describe it in terms of what it does.
- Tailor to the user's expertise, e.g. if a novice asks about a topic, describe the topic in general terms and not in system specific terms.
- Tailor to the domain knowledge, e.g. if a user knows how all the components of a topic works and how the topic is structured, he also knows how the topic works.
7 Task support and tailoring

Explicit task support in MediaDoc is provided by a task agenda component. The user starts a task by selecting a task operation on a topic. For example, in figure 6 the user is viewing a code modification request. Selecting the 'Analyze Modification' option on the task menu for this topic (i.e., Route modification) starts the task.

When the task is started it is pushed to the task list. In the Tasks frame of figure 7 we can see that Perform Analyze Modification on Route modification is pushed to the task list. The user can have more than one task in the task list. Starting a new task will add the new task to the list, clicking on any task in the list will make that task active and clicking on the (X) after the task or selecting stop in the task model frame will delete the task from the task list.

After the task is pushed to the task list, the generic task model for the task is instantiated and placed in the task agenda (i.e., Task Model frame in figure 7). Generic task models capture the user goals, the task steps and the suggested methods for performing these task steps. Part of the generic task model for the Analyze Modification task is given in figure 5. The instantiation of this generic model for the Route modification topic is shown in the Task Model frame of figure 7. Here the first task step is to collect topics of interest. Two methods, namely Assess Topics and Search for Topics is suggested for this task step. In figure 7, the user has already clicked on the Assess Topics method and MediaDoc suggested Fixed Wing Aircraft as a topic the user should look at. Assessment method in MediaDoc is similar to asking a human expert about what to learn or where to start the task. The assess method in MediaDoc decides what topics to suggest to the user by checking which of the relevant topics for the task is not known by the user. In figure 7, Fixed wing aircraft was relevant to the task and after checking the user model MediaDoc decided that the user should learn about it.

The second suggested method for collecting topics is doing a search in the documentation. This is similar to the keyword searches used by the subjects in the task study. The user can do a keyword search and post the results to the task agenda for later investigation. In figure 7, the user has already done a search and posted the topic route to the task agenda. The users can post any topic of interest to the task agenda and can delete them by clicking on the (X) after the topic name.

At any point only one of the task steps is active. The active task step is shown in italics and bold for-}

Collect Topics (goal=collect)

Assess topics (find out what the user does not know about the current topic)

Search topics (search in documentation space)

Ask about topics (goal=comprehend, generate what, self), (how, self) for collected topics

Collect behaviors

Map topics to behaviors

(for each topic in collect topic, find the corresponding behavior)

Search behaviors (search among behaviors)

Figure 5: Part of the generic task model for Analyze Modification task

mat. In figure 7, Search for topics is the active task step. The goals associated with this task step is assumed to be the current user goal. If the current task step doesn’t have an associated goal, then the goal from the parent node will be inherited. So in figure 7 the current user goal is gather. MediaDoc uses the current goal in planning the explanation. For example, in figure 8, the user wanted to get information about Fixed Wing Aircraft. Since the current goal is gather, MediaDoc gave an explanation that shows the relations between the Fixed Wing Aircraft and other topics in the database. All the relations in the database that originate from or end at the Fixed Wing Aircraft topic are listed under the related links. In addition, the finite state machines for every related behavior are also listed. In contrast, in figure 9 the goal is comprehend and the explanation focuses on the comprehension of Fixed Wing Aircraft rather than its relation to other topics.

When the user finishes collecting topics, he will click on the Ask about topics task step. This will generate the questions specified in the generic model for each one of the collected topics. In figure 9, (what, self) and (how, self) questions are generated for each one of the collected topics. The user then selected the What is Fixed Wing Aircraft to see the answer to the question. This produced the tailored answer to the question in the explanation frame. The user can post questions to the task agenda by posting topics and using the Ask operations to generate questions or they can post the questions directly to the task agenda. The recording of the questions in the task agenda helps users in recalling the hypotheses they generated. In addition, it shows the users the remaining hypotheses that needs to be checked.

Finally, figure 10 shows the Collect Behaviors task step. The suggested methods for this task step are Map topics to behaviors and search. Map topics to behaviors method finds all the mappings from the collected topics to the behaviors in the database. In this example, MediaDoc found the Sweep taskframe behavior for the Fixed Wing Aircraft topic. The user can continue finding mappings between different layers as specified in the generic task model and the task agenda.
Modify the behavior of fixed wing aircraft such that the flight path can be changed dynamically during flight.

Figure 6: Starting the route modification task

Figure 7: Search support
Figure 8: Fixed wing aircraft explanation during information gathering

Figure 9: Fixed wing aircraft explanation for comprehension
keeps track of these mappings. In the current version of MediaDoc, the layers of abstractions are explicitly coded in the generic task model and the mappings between the layers are simple database searches. However, one can further generalize the support for the mappings between layers of abstractions by specifying the starting/ending layer of each task and by specifying the mapping operations between different layers.

8 Future Work

Our research goal was to develop an interactive software explanation tool that can act as a system expert and that can produce tailored explanations to user questions which are easy to understand and are relevant to the user goals. The latest version of MediaDoc provides such a tool. Now we are preparing an experiment to evaluate the effectiveness of MediaDoc in software understanding. Using the documentation and the source code provided with the ModSAF system, we prepared two versions of online documentation. The first version is prepared using HTML linking. The ModSAF source code and the documentation is processed and links are created between related documentation pages. The second version is prepared using MediaDoc. It uses the same set of links created for the first version, but adds the semantics of the question model to the links. In addition, this version has the explicit task support.

Our evaluation plan is to assign a task to subjects and have them perform the task using one of the explanation systems. After the task is completed, the subjects will be asked to answer questions that measure their comprehension of ModSAF. We will then compare the two groups of subjects by using the answers they provide and also by evaluating their task performance.

9 Conclusion

Software understanding has become an important problem in the software industry. Better tools are needed for solving this problem and these tools can be developed only after we study how the users understand software. Our research goal is to build a tool that supports human software understanding. We studied user questions within and outside the task context and found out that interactivity, user and task tailoring are important requirements for such a tool. We took a planning approach solution to address these requirements and were successful in producing explanations tailored to the users' expertise and task. We are in the process of evaluating the effectiveness of the tool in understanding large software systems.
10 Acknowledgments

This work is sponsored by DARPA under DARPA order number D880.

References


