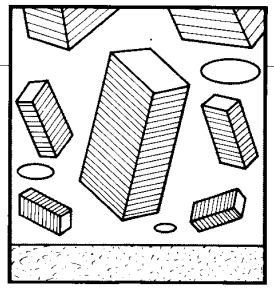

A Hypertext System to Manage Software Life-Cycle Documents

Pankaj K. Garg and **Walt Scacchi**, University of Southern California

Traditional systems don't handle the documentation requirements of large-scale, multiproject software development. But this hypertext-based system does.

Documenting software systems is a necessity, since not all relevant information about the system can be embodied in a system's source code. The larger the system, the more critical the problems of documents' consistency, completeness, traceability, revision control, and retrieval efficiency.

For the next generation of operating systems, Robert Balzer has envisioned an environment[1] based on objects and relationships between objects, as opposed to the conventional file-based systems. Using this philosophy, we have constructed the Documents Integration Facility for the development, use, and maintenance of large-scale systems and their life-cycle documents.

The philosophy underlying DIF is twofold. First, it is necessary for an effective engineering-information system to prescribe the information that must be stored, to ensure the completeness and preciseness of the information. Second, for the engineering-information system to be generally applicable, it should be possible to change the information requirements prescribed by the system according to the needs of different settings.

DIF helps integrate and manage the documents produced and used throughout the life cycle — requirements specifications, functional specifications, architectural designs (structural specifications), detailed designs, source code, testing information, and user and maintenance manuals. DIF supports information management in large systems where there is much natural-language text.

DIF has been used to integrate and manage the life-cycle documents of more than a dozen systems (in the System Factory) totaling more than 200,000 lines of code, developed by more than a dozen teams of three to seven people each. More

---

Garg is now at Hewlett-Packard Laboratories. An earlier version of this article appeared in *Proc. 21st Hawaii Int'l Conf. on System Sciences*, CS Press, Los Alamitos, Calif., 1988, pp. 337-346.

0740-7459/90/0500/0090/$01.00 © 1990 IEEE

than 40 Mbytes of life-cycle descriptions have been managed by DIF. The results of this experiment suggest DIF's potential usefulness for larger systems (those of more than a million lines of code).

In a typical project's life cycle, information about the process and the product is diffused among several individuals, and it is easy both for information to be lost and for information bottlenecks to be created. DIF encourages the easily accessible storage of relevant information about the process and the product.

In DIF, we consider segments of software documents as the objects to be stored, processed, browsed, revised, and reused. Links explain the relationship between the objects. DIF stores the objects in files and the relationships between the objects in a relational database, resulting in a persistent object base. This eases the reuse of documents.

DIF also provides software-engineering tools to process the information in the objects. By judiciously using links, keywords, and information structure, DIF users can alleviate problems of traceability, consistency, and completeness. Through the Unix RCS revision-control facility, DIF provides revision control. Through interfaces to the browsing mechanisms of the Ingres database system, DIF provides efficient document retrieval. Through the Unix mail system, DIF lets project participants exchange structured messages. Through language-directed editors, DIF eases the development of software descriptions in several formal languages. And through software-engineering tools for functional and architectural specifications, DIF aids the analysis of formal descriptions.

Documents in hypertext

Through the development process, DIF stores all relevant information about the target system in textual objects as nodes of hypertext. Hypertext is a storage structure where information is stored in the nodes of a graph. Links between hypertext nodes allow efficient browsing of the information. Attributes attached to nodes and links provide information filtering.

No restriction is put on the nature of information in the nodes, so the same hypertext may contain a node of natural-language text and a node of program code. This is the main advantage that hypertext systems have over conventional database-management systems, document bases, and knowledge bases.

Through the development process, DIF stores all relevant information about the target system in textual objects as nodes of hypertext.

Documentation method

To effectively manage the documents in a large-scale software process, you must first understand what needs to be documented. To this end, we use the System Factory documentation method, which is an eight-year-old laboratory project at the University of Southern California set up to experiment with novel ways of software-project management and with the development, use, and maintenance of innovative software technologies. The uniqueness of the System Factory method of documentation is the blend of organizational and technical concerns that it advocates.

In the System Factory, the software process is broken into activities, where each activity culminates in the production of a document. The eight documents that emerge from the System Factory method are:

- Requirements specification. This document describes both the operational and nonoperational requirements of the system being developed.

Operational (testable) requirements outline the system's performance characteristics, interface constraints, quality-assurance standards, and human factors. The operational requirements are defined so you can trace them through design and implementation.

Nonoperational requirements outline the organizational resources available to support system development, the package of resources being built into the target system, forethoughts about the system's development life cycle, assumptions about the system's operation in its target environment, and expected changes in the system's operational requirements over its lifetime.

The requirements-specification document is written in a natural language, for which DIF provides an interface to text-editing and -formatting systems. DIF provides mechanisms like keyword association and linking that help you browse through the requirements. By defining a form for the requirements document, the manager of several projects can standardize the contents of the requirements document at the level of sections and subsections. By judiciously choosing keywords and links, the manager can set up the requirements to ease the tracing of operational requirements through the system's design and implementation.

- Functional specification. This document details the computational functions the target system will perform in terms of the computation objects, their attributes, attribute-value ranges, object and attribute relationships, the actions that manipulate them, constraints on the actions, global stimulus/response monitors, and the system agents that organize and

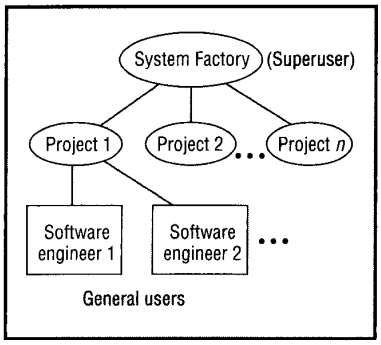

Figure 1. System Factory structure.

embed these into a system environment.

You write these formal functional specifications in the specification language Gist,[1] which promotes the incremental development, refinement, and rapid prototyping of operational system specifications.

In incremental development, you first give an informal, narrative specification. Next, you describe the objects (both active and passive) in the target system and its environment in a graphical form. Finally, you give the formal description of the objects and agents in Gist.

DIF recognizes the formal Gist text and lets you send it for processing through the Gist specifications analyzer and simulator. It also automatically activates the Gist language-directed editor.

- Architectural specification. This document describes the system modules' interconnection structure with defined data-resource interfaces, arranged to facilitate parallel detailed design and implementation. You use the NuMIL module-interconnection language[2] to descibe this document. You can also formally define system-timing and concurrency constraints in this document.

DIF provides access to the NuMIL editor and processing environment, which lets you define modules and their resource dependencies as well as check and track changes in the modules. It also provides access to a structure visualizer that graphically displays the module interconnections.

- Detailed-design specification. This document describes (in Gist and NuMIL) the behavioral algorithms and system-dependent operations consistent with the computational modules and resource interfaces defined in the architectural specification.
- Source-code document. This contains the target system's source code and reflects the system structure as detailed in the earlier documents.
- Testing and quality-assurance document. This document stores test cases that specify how you can trace the operational requirements to test-case runs to validate the system's performance. You can use keywords and links in DIF to link the test cases to the operational requirements.
- User manual. This document describes the commands, error messages, and example uses of the system in a standard user-manual format.
- System-maintenance guide. This document describes how the system can be enhanced, how its performance can be better tuned, known system bugs, and porting constraints.

Coupling this documentation method with DIF lets you produce, organize, and store encyclopedic volumes of life-cycle information for subsequent browsing, reuse, and revision.

Interfaces to Latex, spell checkers, and other tools provide a text-processing environment akin to the Unix documenter's workbench.

DIF structure

Figure 1 shows the organizational structure supported by DIF in the System Factory. In the System Factory, the project manager prescribes what needs to be described in each document. There are potentially several projects in the factory at the same time, and several software engineers work on each project.

Operation modes. Corresponding to the type of users in the System Factory, DIF allows two modes of operations: a superuser mode and a general-user mode. The two modes are comparable to the database administrator and the user, respectively. In superuser mode, you define the factory structure (what projects are in the factory and who is responsible for each one) and the structure of the documents (what needs to be documented). In general-user mode, you create, modify, and browse through the hypertext base. The general user can operate at two levels: the information level and information-structure level.

Forms and basic templates. The superuser defines the forms and the basic templates in the factory. One concern of the System Factory was to ensure that all the projects have a standard document structure. Thus, each document is defined as a *form*, which is a tree-structured organization of basic templates to be instantiated with project-specific information. Figure 2 shows an example requirements-specifications form. Such forms provide a way of defining the process that is to be followed by the project members.

The superuser defines each form only once; *all* projects inherit that form. This allows standardization of documents across projects. It also lets the software engineers concentrate on the *content* of the documents without being bothered about their structure.

When defining the basic templates, the superuser also defines the nature of information that needs to be given in each basic template. This entails providing the type of the basic template (narrative, graphical, NuMIL, Gist, C code, executable code, or object code) and a text template that the general user can use to enter information in the instances for that basic template.

General users can define additional project-specific basic templates, but they cannot modify the basic templates defined by the superuser. Instead, they create instances of the superuser-defined basic templates and enter information in those instances. In a project, a basic template can have many instances.

Project information. The superuser provides project information. Project-related information in DIF consists of a list of projects and the software engineers working on them. This information lets DIF check users' read and write privileges. There is no restriction on who may read the information of any project. The super-

Section number	Section heading
1.	Overview and summary
2.	Problem definition
2.1.	Technology in use
2.2.	System diagram
2.3.	Theory of system operation
2.4.	Intended application
2.5.	User skills
3.	Operational requirements
3.1.	Performance characteristics
3.2.	Standard interfaces
3.3.	Software quality-assurance plans
3.4.	Software portability
3.5.	User orientation
4.	Nonoperational requirements
4.1.	Resources available for development
4.2.	Package of resources built into the system
4.3.	Forethoughts about the system's life cycle
4.4.	Assumptions about system operation
4.5.	Expected changes in operational requirements
Basic-template number	**Basic-template heading**

Figure 2. Requirements-specification form.

user has read and write privileges for all information.

A general user can add links, keywords, and annotations to other users' and other projects' basic templates. (The links, keywords, and annotations are tools that allow easier later browsing of the project information — something we call "information trailblazing.")

Adding more project information like tasks, schedules, and progress reports is a natural extension to DIF hypertext. We have described this issue at length elsewhere.[3]

Information level. DIF facilities let users enter, modify, and use the information required by the forms as dictated by the superuser. DIF provides language-directed editors for all the formal languages used in the System Factory. The general user need not worry about the files that must be created when entering information in basic templates because DIF automatically generates a file name based on the project and the basic template.

For backup, DIF can store entire forms in RCS. DIF lets you check forms in to and out of RCS. Options like retrieving revisions through user-defined identifiers and cut-off dates are available through the DIF interface.

You can request that a software tool process the information in a basic template from within DIF itself, without entering the operating system. For example, if a basic template contains C code, you can request that the code be compiled.

Interfaces to Latex, spell checkers, and other tools provide a text-processing environment akin to the Unix documenter's workbench.

Information-structure level. The information-structure lets the general user navigate through the information hypertext stored in DIF. (This level is different from a database schema because, in a schema, the structure does not change with the information. Here, the structure depends on the currently defined links.) DIF provides these navigation mechanisms:

• Links. The user (either superuser or general user) can define links between basic templates. You can link the system's operational requirements to the code modules that support the capability. Links may be *operational* links, which readily support situations where executable descriptions must be linked to the source code. For example, you can link a C-code basic template to the executable basic template derived from that code. Visiting that operational link results in the execution of the linked basic template. Future versions of DIF will support arbitrary shell-procedure attachments.

DIF supports two types of links: node-to-node links, which are relationships between the definition of two basic templates, and point-to-point links, which link one point in a basic template to a point in another basic template.

• Keywords. For each basic template, you can define keywords that describe the semantics of the information contained in that basic template. DIF stores the keywords associated with basic templates in a relation maintained in the Ingres database. This lets you use the Ingres querying facilities to navigate through documents.

For example, you can look for all basic templates (within and across projects) that have a particular keyword, list the keywords of a basic template, or search for basic templates that have keywords satisfying a pattern. DIF also has standard browsing functions for users not trained in the Ingres mechanisms.

In creating DIF, we were mainly concerned with the efficient storage and usage of keywords, not how they are derived. You can attach automatic keyword-generation tools to DIF if you want.

DIF lets a document's readers create keywords of their own. This lets new personnel in a project team quickly tune the documents to their needs.

• Forms and compositions. Forms are a convenient way to view the documents related to each life-cycle activity.

To fully use the potential of the hypertext information in DIF, you can define your own composition of basic templates. A composition is similar to a form, except that it is not enforced on all projects but is associated with the user who is browsing the documents. You define compositions, as you do forms, by defining the constituent basic templates. You can also define compositions on the basis of the trail you have followed while browsing through the hypertext.

You use compositions to print hardcopy documents, much like the path facility in a typical hypertext system. We have developed special-purpose tools to correctly compose different types of basic templates and to generate appropriate markup code for the Latex text-formatting system. You can also use compositions to generate make files for different system configurations.

Document integration and parallel development. DIF provides several features that let you view system information in an integrated manner within and across projects. The documents are organized in a tree of Unix directories and files, as Figure 3 shows. Although you enter information in DIF at the file level, DIF invisibly handles all the routine file-management functions (like creating directories for related

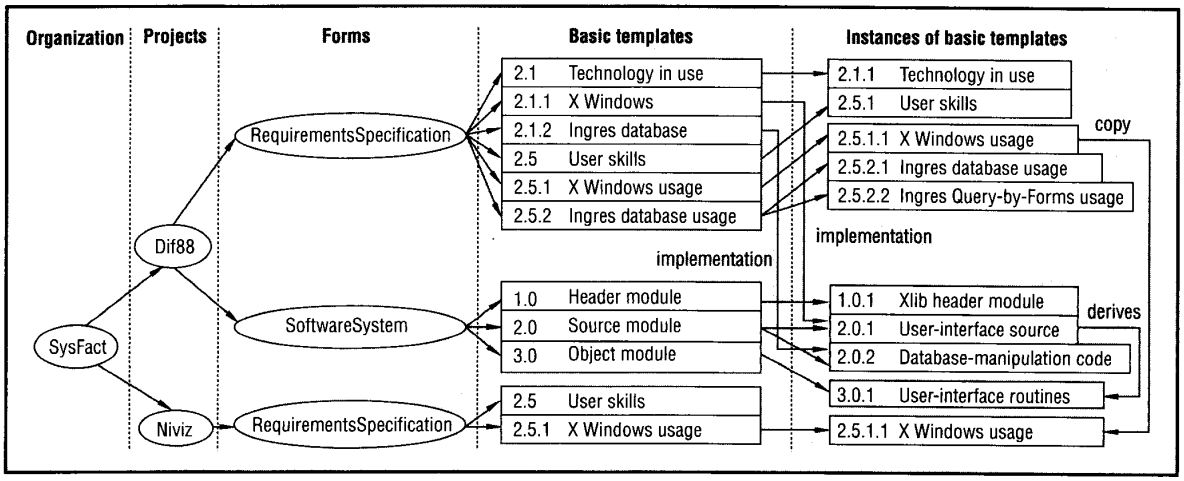

Figure 3. Hypertext structure for life-cycle documents. At the far left is an organization (System Factory) with two projects (Dif88 and Niviz). Each project has forms like Requirement Specifications and Software System. Each form has basic templates defined by the superuser (common to all projects) or general user (specific to one project). Each basic template may have more than one instance in a project. Objects in ovals are mapped to Unix directories, while those in rectangles are mapped into Unix files.

documents).

For example, when entering information for the system's operational requirements, you do not have to create the file to store the text associated with the operational requirements; DIF does it automatically. You need only be concerned with creating or manipulating software descriptions. This provides an object-oriented environment of persistent descriptions rather than simply a loose collection of files and directories.

DIF lets the engineers in the System Factory develop parts of documents in parallel without worrying about integration issues. Thus, one person could be writing the target system's operational requirements while another person is writing its nonoperational requirements. The individual efforts are automatically merged in the same hypertext document.

File structure. At the heart of DIF's implementation are the Unix file system and the Ingres database system. The file system provides a repository for the textual and graphical information; Ingres stores information-structure-level and project-level information. DIF builds a file structure (on Unix) that models the System Factory structure (see Figure 1). Figure 3 shows an example file structure.

At the root of this file structure is the System Factory directory that contains directories for each project. Under every project directory is a directory for each form defined in the factory. Under the directory for each form are files for the basic templates in the form. For example, Figure 3 shows part of the requirements-specifications directory in the System Factory. Files in this directory represent basic templates, like 2.1 (Technology in Use) and 2.5 (User Skills).

User-defined instances of basic templates are also maintained as files. For example, in Figure 3's Dif88 project, basic template 2.0 (Source Module) has at least two instances: User-Interface Code and Database-Manipulation Code.

Using DIF

Figure 4 shows DIF's top-level interface, which shows a snapshot of the system with the user editing three basic templates. The workstation screen is divided into three main window types: command, status, and workspace. You issue commands to DIF by clicking one of the command windows. DIF handles textual I/O through pop-up windows. The status window shows information like the current project, the number of basic templates visited in this session, the version numbers of the systems being used, the current user, and the current mode. DIF creates all browsing windows in the workspace window.

In Figure 4, Editor Window 1 shows an annotation example. The basic template (2.5 User Skills) in Editor Window 2 is linked to the basic template (2.5.1 X Windows) in Editor Window 3 with the link ElaboratedIn. Editor Window 3 was opened by clicking the Target File part of the link window that in turn was obtained by clicking the link icon in Editor Window 2.

Hypertext editing. We have extended the X Windows editor to provide two kinds of hypertext functions: the capabilities to annotate the text at any point and to link two points in two basic templates.

To add an annotation at a point in the text, you place the mouse pointer at the point and press F2. This opens a dialogue window with an editor window in which you can enter an annotation.

In the text, annotations are marked with a special character icon. Any user can look up an annotation by placing the mouse at this icon and pressing the F1 key. You can have multiple annotations at a single point; this supports group discussions. DIF automatically records attributes like the annotation's creation date and author.

To add a link from one basic template to another, you open the two templates in two separate editor windows. You then position the mouse pointer at the link's source position and press F3. DIF asks for the link name to be added. After entering the name, you position the mouse pointer at the second template and press F4. As with annotations, DIF displays links in the text as special character icons (the icon is different from the annotation icon).

To display the details of a link, you position the mouse at the link icon and press F5. To follow a link, you click the Target File item in this display of link details; DIF then opens another window that contains an editor for the link target. You can associate link names with special-purpose scripts that will be executed when the link is traced.

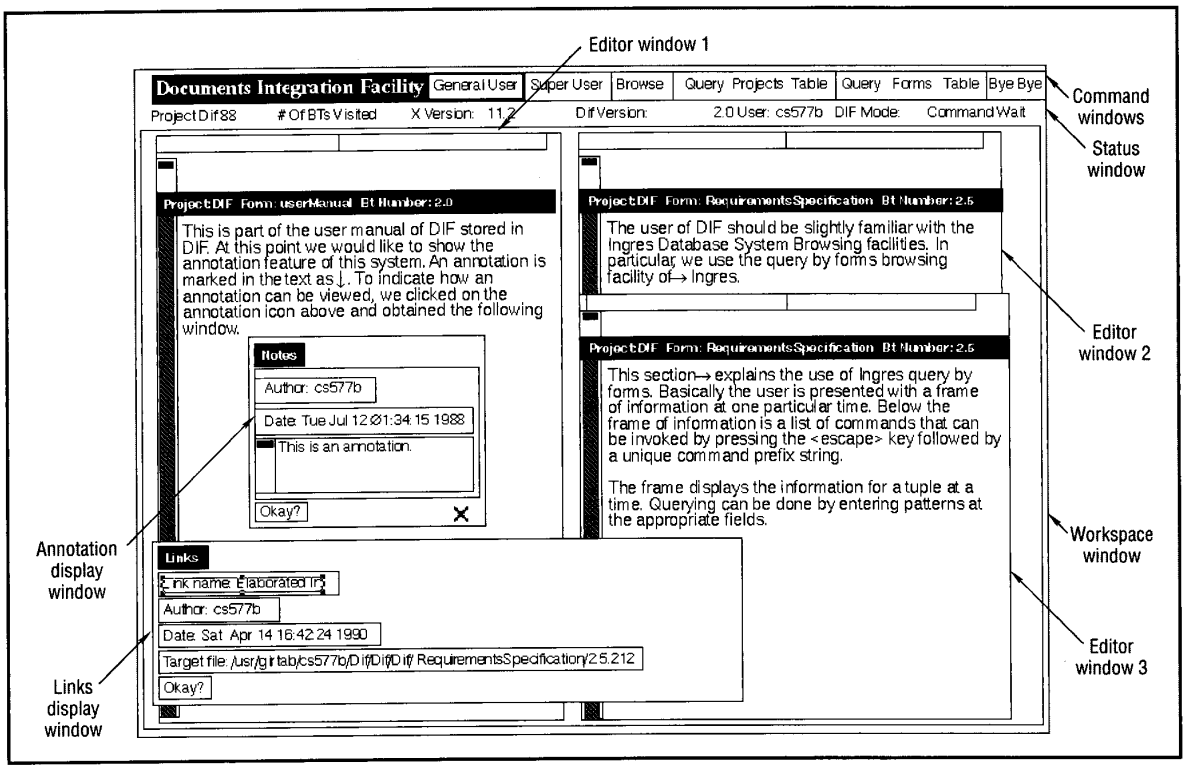

Figure 4. DIF screen layout.

Command modes. There are six modes available in DIF's command-windows title bar: general-user, superuser, browse, projects-table query, form-table query, and session-end (the Bye Bye window).

General-user mode. By clicking the general-user command window, you get a menu as shown in Figure 5a. This provides functions at the general-user level like creating or deleting an instance of a basic template, creating or deleting a project-specific basic template, editing the information contained in a basic template, processing the information in a basic template through a software-engineering tool, mailing a basic template to another project participant, and various operations on compositions of basic templates.

To edit a basic template, you select the Edit Bt item from the menu. DIF presents a menu of available forms and basic templates. Once you select the appropriate form and its constituent basic template, DIF opens a window with an editor process to edit the corresponding file. This reflects the buffering that DIF provides

Create an Instance of a BT	Add Project	Browse Links Table
Delete an Instance of a BT	Delete Project	Browse Keywords Table
Create a BT	Rename Project	List Keywords of Current Project
Delete a BT	Add Project Users	List Keywords of Another Project
Edit a BT	Delete Project Users	List BTs having a Keyword, within project
Invoke a Tool	Create Form	List BTs having a Keyword, across projects
Define a Composition	Delete Form	List Forms having a Keyword, within project
Add a BT to a Composition	Delete BT	List Forms having a Keyword, across projects
Delete a BT from a Composition	Add BT	List Keywords of a Compositions
Show a Composition Definition	Change Password	List Projects having a Keyword
Print a Composition	Clear Database	Step through Links of a BT
Delete a Composition	Print a Screen Dump	HELP
Change Project	HELP	
HELP		
(a)	**(b)**	**(c)**

Figure 5. Menus for **(a)** general-user, **(b)** superuser, and **(c)** browse functions.

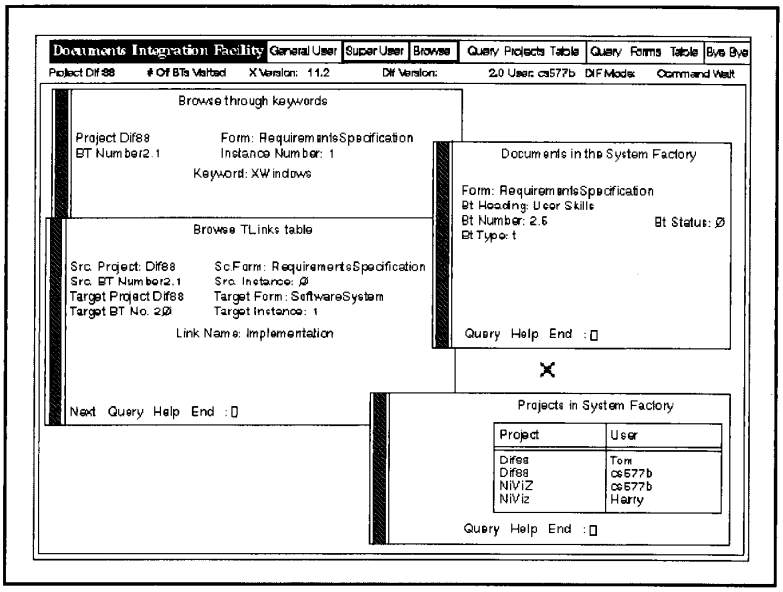

Figure 6. Browsing through Query Forms Table.

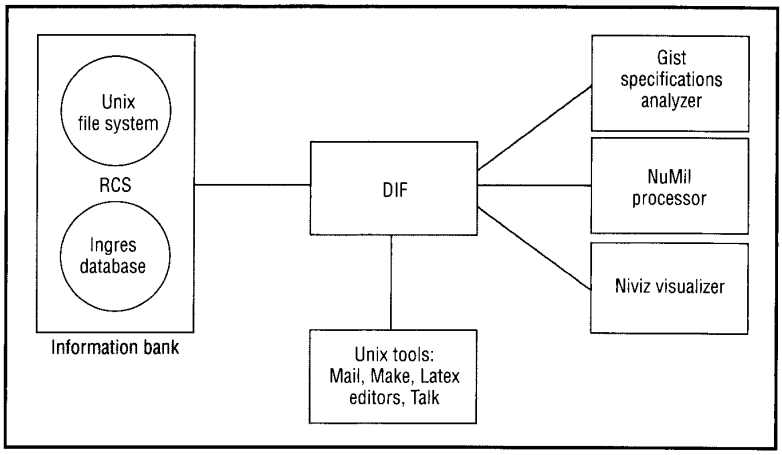

Figure 7. DIF's organization.

from the operating system's file-management facilities. You can change the default editor through a parameter supplied to DIF on its invocation. While you are editing a basic template, you can choose to edit another basic template in another window by either the same selection mechanism or by following a link from any of the previous basic templates being edited.

Superuser mode. By clicking the superuser command window, you are presented with the menu shown in Figure 5b that lets you act as the administrator of the hypertext document, providing functions to create, modify, and delete forms, projects, and project members. When defining forms, the superuser can define the process model that is to be followed in the life cycle of the various projects.

Browsing. The next three command windows — Browse, Query Projects Table, and Query Forms Table — provide browsing facilities for the hypertext storage structure. The Query Projects Table and Query Forms Table command windows provide interfaces to Ingres query-by-forms facility, which gives you access to several methods to make queries about the projects and the forms' structure.

Figure 6 shows a typical usage of this facility. The windows in the workspace were created per user requests. For example, when you click the Query Projects Table command, the window labeled Projects In The System Factory opens. In this window, you can query the database about the projects in the System Factory by specifying patterns for either projects or users. The position and sizes of the windows are always under user control, a feature provided by X Windows.

The Browse command window provides many predefined browsing facilities for the hypertext document. Figure 5c shows the menu of these facilities. You can create, delete, or browse through links between basic templates. Using keywords, you can search for appropriate projects, forms, compositions, or basic templates.

DIF environment

Because DIF can accommodate all life-cycle activities, we consider it to be a software-engineering environment. With the progress of the target system through the various life-cycle activities, DIF offers a uniform interface to access the appropriate tools, like a functional specification analyzer and an architectural design processor, at each phase. In the interfaces to these tools, it supports the notion of interface *transparency*: DIF provides unobtrusive use of the tool that it interfaces to while providing mechanisms that the tool itself lacks.

Figure 7 shows the organization of DIF with respect to the other tools in the System Factory. The basic set of tools, described in detail elsewhere,[4] contains

- a Gist specification analyzer and simulator that aids the development and use of the formal functional specifications of (sub)systems under development,
- a module-interconnection and interface-definition processor that supports the design and evolution of multiversion system configurations described in the NuMIL language,
- an Emacs-like language-directed editing environment for constructing and revising structured documents and system description languages like Gist, NuMIL, and C,
- a systems visualizer that graphically presents system configurations, and
- Unix tools like RCS, Make, Spell, Nroff/Troff, Talk, and Mail. The interface to the mailing system helps people coordinate their activities via structured messages consisting of basic templates.

Design issues

In designing DIF, several issues concerned us, including tools interface, consistency and completeness, reuse, scaling up, on-line information, multiproject support, and revision control:

• Tools interface. Although the Unix philosophy of providing loosely coupled tools and using them either independently or coupled through piping mechanisms has been fruitful, the plethora of tools now commonly available has thwarted their efficient use. Developers need a structured way to access the tools. Systems like DIF that organize the tools according to their usage will improve the utility of the underlying tools and thus software engineers' productivity. The ease of adding tools to DIF is also an advantage to engineers.

• Consistency, completeness, and traceability. A natural concern for document management is to provide mechanisms that help maintain documents' consistency and completeness over time and that provide traceability mechanisms across different life-cycle documents.

DIF addresses consistency through its keyword and link mechanisms. You can use keywords to locate related documents to propagate changes. You can use automated links to define how changes in the source basic template should be propagated to the target basic template.

DIF addresses completeness through the notion of forms for defining the life-cycle documents. They let the superuser ensure that critical system aspects have been documented at the appropriate places.

DIF addresses traceability through the use of links and of keyword-searching and navigation mechanisms.

• Reuse. System documentation is a time-consuming process, and it is imperative that we find ways to reuse documents. DIF is a first step in this direction, since it provides a hypertext-based persistent repository of document objects with facilities for efficient browsing and retrieval. Thus, you can locate and reuse basic templates. In an empirical study done in the System Factory, we found DIF to be used frequently to retrieve and convert old documents into new ones, especially when the old documents were from a closely related project.

• Scaling up. To support large-scale systems engineering, we tried to choose wisely between what information goes into the database and what information goes into the file system. Most of the textual information (except keywords) is stored in the Unix file system. In fact, we exploited the Unix file system's structural mechanisms to encode some of the documents' structural information.

The information stored in the database is therefore minimal. Throughout the System Factory experiment, we have found that the size of information in the database system is no greater than 1 percent of the information in the file system. This requires further experimentation but if the results are at all indicative, DIF has been able to achieve the kind of storage distribution that we wanted — and it did so with existing file and database systems.

• On-line information. One thrust of the DIF philosophy is to shift the medium of information exchange in the software process from paper to on-line form. This has the obvious advantage of being a dynamic medium, since documents can be modified very easily. On the other hand, paper documents have advantages like physical feel, the ability to be written on, and ability to be read almost anywhere. DIF has covered both bases by providing

Other hypertext systems

DIF is in some ways similar to current hypertext systems, including commercial systems like Guide[1] and research projects like Sodos,[2] Textnet,[3] Planetext,[4] Notecards,[5] and Neptune.[6] All share the goal of managing textual information.

But DIF differs from most other hypertext systems in that it is geared toward facilitating information management in the software process. It can define the activities of a software process so that you can easily produce, use, and revise the appropriate information.

We have integrated DIF with several tools in the System Factory to provide an integrated software-engineering environment. For example, you can enter a system's functional specifications through a Gist editor that is syntactically tuned for the Gist specification language. Through an interface with RCS, DIF provides a revision-management facility. As such, DIF can serve as a basis for configuring several software-engineering (CASE) tools. The other hypertext systems cannot, since they are mostly closed systems.

DIF lets you store the documentation for multiple projects in the same hypertext base. It allows manipulation of links between documents across and within projects. It maintains project-user information for access control. Keywords associated with the nodes in the hypertext let you browse documents within and across projects.

The project most similar to DIF is the Sodos system designed by Ellis Horowitz and Ronald Williamson. Sodos was implemented in Smalltalk and, like DIF, managed life-cycle documents in an object-oriented fashion. But key differences set the two systems apart: Sodos manages documents for single projects only. Its revision mechanism is limited to letting you define the revision numbers for parts of the documents stored. Sodos was not built as part of a software-engineering environment and thus did not provide interfaces to any development tools.

Textnet, Notecards, and Planetext support the notion of building hypertext systems to support the management of textual information. Because they are not oriented toward software documents, their concern for relevance and completeness of information, maintaining different versions concurrently, and providing access to other tools to process the information contained in the nodes is minimal.

The Dynamic Design environment built on top of Neptune is designed as an engineering-information system. The concepts of basic templates and compositions in DIF are similar to the concepts of project components and contexts in Dynamic Design. However, Dynamic Design does not support multiple projects and does not support concepts similar to forms and instances of basic templates, which are very important in DIF.

References

1. P.J. Brown, "Presenting Information on Workstation Screens," in *Workstation and Publication Systems*, R.A. Earnshaw, ed., Springer-Verlag, New York, 1987, pp. 122-128.
2. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Sept. 1987, pp. 17-41.
3. R.H. Trigg and M. Weiser, "Textnet: A Network-Based Approach to Text Handling," *ACM Trans. Office Information Systems*, Jan. 1986, pp. 1-23.
4. J. Bigelow, "Hypertext and CASE," *IEEE Software*, March 1988, pp. 23-27.
5. F.G. Halasz, "Reflections on Notecards: Seven Issues for the Next-Generation Hypermedia Systems," *Comm. ACM*, July 1988, pp. 836-855.
6. E. Horowitz and R. Williamson, "Sodos: A Software-Documentation Support Environment: Its Definition," *IEEE Trans. Software Eng.*, Aug. 1986, pp. 849-859.

sophisticated mechanisms for hard-copy rendition of documents and hypertext mechanisms for on-line text management. With DIF, you can potentially get the best of both worlds by not having to worry about storing paper documents and at the same time being able to use paper documents when convenient.

• Multiproject support. DIF is unique among the current hypertext systems for software documents in that it provides support for multiple projects through the same hypertext base. (The box on p. 97 describes some other hypertext-based systems.) This is an advantage in large project environments where related subsystems are developed almost independently and the efforts must later be merged. Through the use of appropriate attribute links and keywords, you can use the information-structure-level information in DIF to tie together otherwise independently developing projects.

• Revisions. The interface to the RCS revision-management facility lets you manage revisions on complete forms. Only one revision of a form is active at a time, and the database reflects the information structure for that form. However, DIF does not provide mechanisms to manage revisions of the information in the database. This means that the information in the file system has revision trees but the database system has no such corresponding structure. We are planning to remedy this deficiency by offering revision management for both the database and file systems in future versions of DIF.

Experience with the prototype usage in the System Factory has convinced us of DIF's utility. DIF, as an active medium, lets you capture much of the information related to a system's design, development, use, and maintenance. You can change the process model built into DIF. You can easily manage several project documents and even exchange information across projects.

But DIF is not complete. We are now investigating the issues of incorporating into the system more knowledge about the activities that the participants in a project perform. The granularity at which DIF considers activities is very coarse; for example, as requirements specifications and functional specifications. If we could determine finer grained actions, we could support them better.

For example, after defining a new requirement, the user of the target system normally wants to communicate the definition to the developers. If this knowledge is precoded in the support environment, the communication act does not have to be carried out explicitly by the users but can be carried out by the support environment as soon as the requirement is defined. Similarly, the users can subscribe to events. For example, if someone has subscribed to the event Change Bt *X*, every time someone else changes basic template *X*, that person can be informed of the change. Our group is pursuing efforts in this direction. ❖

References

1. R. Balzer, "Living in the Next-Generation Operating System," *IEEE Software*, Nov. 1987, pp. 77-85.
2. K. Naryanaswamy and W. Scacchi, "Maintaining Configurations of Evolving Software Systems," *IEEE Trans. Software Eng.*, March 1987, pp. 324-334.
3. P. Garg and W. Scacchi, "Ishys: Designing an Intelligent Hypertext System," *IEEE Expert*, Fall 1989, pp. 52-62.
4. W. Scacchi, "The System Factory Approach to Large-Scale Software Engineering," *Proc. MCC University Research Symp.*, Microelectronics and Computer Technology Corp., Austin, Texas, 1987.

Acknowledgments

We thank Salah Bendifallah for extensive comments on earlier versions of this article and the design of DIF. Comments from the anonymous reviewers have clarified some misconceptions and helped improve the presentation. We thank Amitabh Agrawal and John Leggett for their comments on the article. We thank Abdulaziz Jazzar for his contributions to DIF. We acknowledge the contribution of all those people who helped develop and use DIF in the system-factory class at USC in the 1985-86 and 1986-87 academic years.

The work reported here has been supported by AT&T through research grants and contracts, Hughes Radar Systems Group under contract KSR576195-SN8, Pacific Bell, and Eastman Kodak. Garg was also supported in part by the USC graduate school through the All-University Predoctoral Merit Fellowship.

Pankaj K. Garg is a member of the technical staff at Hewlett-Packard Laboratories. He was at the University of Southern California as a student and research associate when conducting the work described here. His research interests include artificial intelligence, hypertext systems, and software engineering.

Garg received a PhD in computer science from the University of Southern California and a bachelor of technology in computer science from the Indian Institute of Technology in Kanpur.

Walt Scacchi is a member of the computer-science faculty at the University of Southern California and director of the System Factory project there. His research interests include very-large-scale software engineering, knowledge-based systems supporting software development, and organizational analysis of system-development projects.

Scacchi received a BA in mathematics and BS in computer science from California State University at Fullerton and a PhD in information and computer science from the University of California at Irvine. He is a member of the IEEE, ACM, American Association for Artificial Intelligence, and Society for the History of Technology.

Address questions about this article to Garg at Hewlett-Packard Laboratories, 1501 Page Mill Rd., Palo Alto, CA 94303; Internet garg@ hplabs.hp.com.