Testing for quality assurance of hypertext applications

Lingzi Jin, Hong Zhu, P. Hall

Department of Computing, The Open University, Milton Keynes, MK7 6AA, UK

Abstract

Testing is a time consuming and expensive activity in the development of hypertext applications. The management of the test process must ensure that the requirements of testing are satisfied but at a minimum cost. Test adequacy criteria can provide objective measures to the extent that a test meets the testing requirements. However, existing software test adequacy criteria are not directly applicable to the test of hypertext applications.

This paper discusses three levels of the requirements of testing hypertext applications. The first level is the validation of the information contained in each node of the hypertext and the correctness of its display on the screen. The second is the verification of the correctness of implementing links between the nodes. The third is the evaluation of system structures for testing usability. For each level of the requirements, test adequacy criteria are proposed. A software testing tool is developed to support efficient testing of hypertext applications. The paper also gives the results of a case study of the testing tool.

1. INTRODUCTION

1.1. Motivation

Recent years have seen rapid growth of interest in hypertext [1]. A hypertext application is a network of nodes of textual and graphical information, which can be interactively displayed, browsed and searched through following links between the nodes. We will distinguish hypertext tools from hypertext applications. Hypertext tools are systems like HyperCard, and many others surveyed by Conklin [1], which enable hypertext applications to be built. Being a powerful tool of human-computer communication, many hypertext applications have emerged in the IT market. Typical application areas of hypertext include computer aided learning, and on-line documentation systems.

As with all other IT products, hypertext applications must be thoroughly tested before being released into the market. Testing a hypertext application is a complicated process, which requires:

1. examining the structure of the network to detect design faults,
2. exercising the linkage between the nodes to check the correctness of implementation,
3. checking the details of the information displayed on the screen to ensure the validity of the information, and
4. evaluating the usability of the system.

It is time consuming and expensive. The management of the testing process must assure that test requirements are satisfied but at a minimum cost. As with all software testing, test adequacy criteria can provide objective and accurate measurements of test adequacy and provide guidelines for the management of testing processes.

Test adequacy criteria are rules which decide whether enough testing has been performed so testing can stop [2]. There are many test adequacy criteria proposed for testing programs [3]. Among the most well known are statement coverage, branch coverage and path coverage. Statement coverage requires all the statements in a program are executed during testing. Branch coverage requires all the control transfers are exercised, while path coverage requires all the paths in the program are executed during testing. Most test adequacy criteria proposed in the literature are based on the program under test and/or the functional specification of software. They usually rely on the characteristics of programming languages, hence, are not directly applicable to the test of hypertext applications. We have not found any adequacy criteria which meet the requirements of testing hypertext applications.

In this paper, we will discuss the basic requirements of testing hypertext applications and propose test adequacy criteria which can meet the requirements. Methods to achieve adequate tests and to reduce test costs will also be given in the paper.

1.2. A graph model of hypertext applications

A hypertext application has the following common features:

• It consists of a network of nodes, which usually contain textual, graphical and other forms of data.

• Each node in the network corresponds to a window displayed on the screen on a one-to-one basis. The linkages among the nodes are supported by window operations in the form of link icons, or buttons. Each button contains a short textual field to indicate the node it points to. Clicking on a button with the mouse causes the referenced node immediately to be displayed on the screen.

• The information contained in the network can be interactively retrieved in four ways: (1) by following links and opening windows successively to examine their contents, (2) by browsing the information following a fixed browse sequence, (3) by searching for a string, keyword or attribute value, and (4) by navigating around the hypertext using a map which shows the structure of the system and where the currently displayed node is.

As shown in figure 1, a hypertext application can be modelled by a directed graph $G=\langle N, A \rangle$, where $N$ is a finite set of nodes and $A \subseteq N \times N$ is a finite set of directed arcs. The nodes of a graph represent the nodes in the hypertext network. The arcs of the graph represent the links between the nodes. We assume that there is an entry node and all the nodes can be reached from
the entry node through a path following a sequence of arcs. The entry node of the graph in Figure 1 is node a.

![Directed graph model of a hypertext application.](image)

**Figure 1. Directed graph model of a hypertext application.**

### 2. TEST REQUIREMENTS

We identify three levels of test requirements for hypertext applications. The first level is to validate the correctness of the information contained in each node and its appearance and layout on the screen. The second level is to check the correctness of the implementation of the links between the nodes. Finally, the highest level is to examine the structure of the whole network for the evaluation of the usability of the system.

In this section, we will discuss the test requirements at each level, propose adequacy criteria to meet each requirement.

#### 2.1. Validation of the Contents and Appearance of Nodes

The most fundamental elements of a hypertext application are the nodes of the network. They contain the basic information. As discussed in section 1.2, the information in a node is displayed on the screen by a window. Therefore, to validate the correctness of the information in a node, it is necessary and sufficient to check the window for displaying the node. Such check must also include the appearance and layout on the screen.

This requires the test adequacy criterion of covering all the nodes of a hypertext by testing. A browse through the hypertext following the browse sequence will satisfy this criterion, provided that the browse sequence covers all the nodes in the hypertext. Therefore, we have the node coverage criterion:

**Node coverage criterion:**

A test of hypertext is node adequate if all the nodes are checked during testing. The node coverage of a test of hypertext is the percentage of the nodes checked by the test.

#### 2.2. Validation of the Implementation of Links

The linkage among the nodes provides the paths for interactively searching for information. Such links are implemented by the button/window display association.

A possible implementation fault is that the button/display association does not coincide with the intended link between the nodes. There is no other
Software Quality Management

A way to test the correctness of this implementation issue, but to press every button once. This leads to the following test adequacy criterion.

**Link coverage criterion:**

A test of a hypertext application is link adequate if all the links between nodes are exercised during testing. The link coverage of a test of hypertext application is the percentage of the links exercised by the test.

### 2.3. Regression Testing of Localised Hypertext

Currently most hypertext is developed with its interaction language in English. But 90% of the world's population doesn't speak English. For hypertext applications to be successful for users who are not English speakers, it is necessary to change them to meet requirements of local languages and cultural customs. This process of adjusting software from its originating country to another country has two aspects, called *localisation* and *internationalisation*. Localisation is the process of converting the existing software into the locale, i.e., local language, culture, and administrative conventions. Internationalisation is the process of preparing the software so that localisation becomes less difficult.

When a hypertext is localised, it must be tested again. This testing is regression testing. In general, regression testing refers to the testing after the modification of software in the maintenance phase of the lifecycle. The testing requirements of a localised hypertext application also include the validation of the contents and appearance of each node and the validation of the implementation of links as discussed in the previous two subsections. Therefore, the node coverage criterion and the link coverage criterion also apply to such testing.

Since the localisation and internationalisation of a hypertext application usually do not change the linkage structure, the test suite used to test the original hypertext application can often be reused to test the localised version. Such regression testing can often be performed rather efficiently. Moreover, if the linkage is unchanged during localisation, it is possible to compare the linkage of the original with that of the localised version to find linkage implementation errors.

### 2.4. Testing Usability

Generally speaking, the usability of a software system includes the following key aspects [6]:

1. learnability, which refers to the time and effort required to reach a specified level of user performance,
2. throughput of the tasks accomplished by experienced users, that is, the speed of task execution and the error made,
3. flexibility, which is the extent to which users can adapt a system to new ways of interaction as they become more experienced, and
4. the positive attitude engendered in the users by the system.

There are a number of ways to evaluate the usability of a hypertext application [6-8], such as:
(1) **analytic evaluation**, which uses formal or semi-formal descriptions to predict user performance,
(2) **observation evaluation**, which involves observing users' behaviour while they are using the system,
(3) **survey evaluation**, which elicits users' subjective opinions of a hypertext,
(4) **expert evaluation**, which involves experts in assessing an interface, and
(5) **experimental evaluation**, which uses scientific experimental practice to test hypotheses about the use of the hypertext.

Experimental evaluation has the advantages that it is replicatable and can provide objective and quantitative data for statistical analysis. It can compare different groups of users. The results are usually reliable. Given the limitation of the resources, one of the crucial issues in the design of an experimental evaluation is how to reduce the extent of the artificiality. This can be achieved by adequate tests. The following will focus on the adequacy criteria for this purpose.

For hypertext applications, one of the most important factors which affect the usability is the easiness of navigation. It has a profound impact on the easiness for the user to find the information they want. Many researchers have investigated various structures of the network which may ease the hypertext navigation problem [9-11]. Figure 2 gives some of such linkage patterns [11].

For a simple pattern of linkage, novice users can learn the structure of the network easily and quickly. Experienced users can also find their ways easily. Hence, the hypertext may have a good learnability and throughput. However, not every hypertext applications can be naturally put into a simple pattern of linkage.

To systematically test the design of the structure of a hypertext for checking the easiness of navigation, the tester needs to follow a path in the network starting from the entry node to see if it naturally leads to the node(s) which contains the information that users may want. The linkage coverage criterion discussed in the previous section is obviously insufficient for this purpose. A collection of paths must be tested. However, a large hypertext application may contain too many paths to be completely tested.

As a trade-off between the completeness and the test cost, we propose the independent path coverage criterion for testing a hypertext application. The basic idea behind the criterion is that a test is considered as adequate if every path in the graph is a linear combination of the paths which have been tested.

A path is a sequence of nodes in the graph and each node is connected to the next by an arc. In this section, we will assume paths starting from the entry node of the graph. A path $P$ is a linear combination of paths $P_1, P_2, \ldots, P_k$, if $P = n_1 \times P_1 + n_2 \times P_2 + \ldots + n_k \times P_k$, where $n_1, n_2, \ldots, n_k$ are integers. Informally, $P = P_1 + P_2$ means that $P$ is the path which contains the arcs in $P_1$ and $P_2$, where an arc may occur more than once and in such cases the path passes through the arc more than once. Here, $n \times P$ means the addition of $P$ to itself for $n$ times. $P = P_1 - P_2$ means that $P$ is the path which contains the arcs in $P_1$ subtracted by the arcs in $P_2$. A formal definition of the notion of linear combinations of paths can be found in [13] and [14].
Considering each test path represents a test scenario, a linear combination of a set of paths, therefore, represents a scenario which is a combination of the scenarios represented by the test paths. Once all the independent scenarios are tested, we can infer the behaviour and performance of the system from combined scenarios and draw the conclusion about the system confidently. Therefore, only those independent paths need to be tested. On the other hand, if a large number of independent paths are not tested, the behaviour of the system on a large number of independent scenarios will be unknown. This leads to the following test adequacy criterion.

**Independent path coverage criterion:**

A test of a hypertext application is independent path coverage adequate, if it contains at least V independent test paths, where V is the maximum number of independent paths in the directed graph model of the hypertext. The number \( v/V \) is called the independent path coverage, where \( v \) is the number of independent paths contained in the test paths.

To calculate the number of independent paths in a given graph, the following theorem of graph theory can be used [9].

**Theorem.** For any graph G, the maximum number of independent cycles equals to

\[
\nu(G) = m - n + k
\]
where \( v(G) \) denotes the cyclomatic number of graph \( G \), \( n \) is the number of vertices in \( G \), \( m \) is the number of arcs, and \( k \) is the number of strongly connected components of the graph.

Notice that, a strongly connected component is a subgraph that for every pair \( (a,b) \) of nodes in the subgraph, there are paths from \( a \) to \( b \) and from \( b \) to \( a \). Therefore, the theorem may not be directly used in testing a hypertext because the number of strongly connected components of the hypertext may not be obvious. To apply the theorem, we add arcs pointing to the entry node from the nodes which have no paths going back to the entry node, then calculate the cyclomatic number of the modified graph, which has only one strongly connected component. This number is the maximum number of independent paths.

As an example, table 1 gives the number of independent paths for each of the six linkage patterns shown in Figure 2. It is worth noting that the number of independent paths of the linkage patterns increases as the navigation becomes more and more difficult. This coincides with our intuition that more complicated structures need more testing.

<table>
<thead>
<tr>
<th>Linkage pattern</th>
<th>Parameters</th>
<th>No. of independent paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>( L = \text{number of leaf nodes} )</td>
<td>1</td>
</tr>
<tr>
<td>hierarchical</td>
<td>( L = \text{number of leaf nodes} )</td>
<td>( L )</td>
</tr>
<tr>
<td>hypercube</td>
<td>( C = \text{No. of columns}, R = \text{number of rows} )</td>
<td>( C*R - C - R + 2 )</td>
</tr>
<tr>
<td>hypertorus</td>
<td>( C = \text{No. of columns}, R = \text{number of rows} )</td>
<td>( C * R + 1 )</td>
</tr>
<tr>
<td>directed</td>
<td>( m = \text{number of arcs}, n = \text{number of nodes}, L = \text{number of leaf nodes} )</td>
<td>( m - n + L + 1 )</td>
</tr>
</tbody>
</table>

3. THE TESTING TOOL HyperTest.

A large hypertext application could include hundreds of nodes and each node may have up to dozens of links. When testing such a hypertext application manually, a tester often finds the network is a maze and spends a lot of time on recording which links or paths have been covered and on finding an untested link or path. It is desirable that a testing tool could lead the tester following a test path step by step so that the tester can concentrate on checking what is displayed on the screen. In this section, we describe such a testing tool, called HyperTest, for the testing of hypertext applications.

3.1. Structure of the Tool.

Figure 3 below illustrates the architecture of a testing tool HyperTest for testing hypertext applications whose text sources are in Rich Text Format (RTF) files.
In addition to the human-computer interface, the tool has the following main components.

1. A source code scanner scans the text source in RTF files to generate the connection matrix which represents the linkage of the hypertext application, and to obtain interface information like browse sequence numbers, field keywords, etc.

2. A test suite generator generates a set of test paths based on the connection matrix. Two algorithms have been developed to generate test suites to meet the link coverage criterion and the independent path coverage criterion.

3. A test executive accepts a test suite as input, and gives instructions to the tester by asking him to press certain buttons or click on certain text fields. It helps the tester execute the hypertext application step by step. At each step, the tester should check and validate the details displayed on the screen.

4. A static analyser, which is designed specially to help the regression testing of hypertext applications after localisation. As discussed in section 2.3., the regression testing of a localised hypertext application is required to verify that the linkage structure of a localised hypertext application is identical to the structure of the original software. This of course can be achieved by testing the jumps all over again, but a more efficient approach is to statically analyse and compare the structures. These functions are fulfilled by the static analyser. Based on the connection matrices built by the source code scanner, the static analyser compares the matrix with the matrix of the original hypertext applications. The result of the comparison is reported to the developer to locate any inconsistency.

It is worth noting that the testing tool HyperTest is independent of the local languages of the hypertext application. It can be applied to hypertext applications in English as well as in French, or Germany, etc.

Another support provided by the testing tool is that the test paths generated to meet the test adequacy criteria can be reused in regression testing.

3.2. Generation of minimal test paths covering all edges.

One of the most difficult issues in testing hypertext applications is the complexity of the network. To satisfy the linkage coverage criterion, the tester must traverse the directed graph to cover all the arcs. However, the network is
Software Quality Management

often a complicated graph rather than a simple tree. It is almost impossible to find a single path which covers all the arcs once and only once without wandering too long in the "maze", and there may well not be such a path[4]. In this section, different from the test paths for testing usability, a test path for testing the correctness of linkage implementation need not start with the entry node.

When the tester covers all the arcs by walking through N paths \(<n_{i1}, n_{i2}, ..., n_{im}>\), i=1,2,..., N, the cost of the testing is the sum of the time to find the node \(n_{ij}\), and the time to follow the links from \(n_{ij}\) to \(n_{ij+1}\). Assume that the hypertext system supports the search of a node by a keyword or a string, the cost to find a node should not be much higher than that to press a button. Therefore, the minimum cost of an adequate test is the number of arcs plus the minimum number of paths which cover all the arcs. By graph theory [5], we have the following theorem.

Let \(\text{In}(n)\) be the in-degree of node \(n\), i.e. the number of arcs leading to the node, \(\text{Out}(n)\) be the out-degree of node \(n\), i.e. the number of arcs coming out the node.

**Theorem.** Let \(G=<N,A>\) be a connected directed graph. The minimum number of paths covering all the arcs, written as \(PN(G)\), is:

\[\sum_{n \in N, \text{In}(n)<\text{Out}(n)} \text{Out}(n) - \text{In}(n),\]

if there is a node \(n\) such that \(\text{In}(n)<\text{Out}(n)\). Otherwise, \(PN(G)=1\). The number will be called the **number of covering paths**.

The algorithm used in HyperTest will generate a set of \(PN(G)\) paths covering all the arcs once and only once. Therefore, such test suite is of the minimal test cost. For example, the path set generated by applying the algorithm to the example in Figure 1 is

\{ \(<a, b, c, b>\), \(<a, e, c, g, e, f, c>\), \(<a, d, f>\), \(<d, e>\) \}.

Readers are referred to [14] for the details of the algorithm.

### 3.3. Generation of a complete set of independent paths.

To satisfy the independent path coverage criterion, a complete set of independent paths must be generated. Such a set of independent paths is called a base path set of the graph, which has the feature that every path in the graph is a linear combination of the paths in the set. A graph may have more than one base set. For simple linkage patterns like hierarchical and directed acyclic graph, it may be very easy to find a base set. For example, the base set for a tree, i.e. hierarchical linkage pattern, consists of all the paths from the root, i.e. the entry node, to the leaf nodes. But, in general, it could be very difficult to find a base set manually. An algorithm to identify the base paths has also been developed. Readers are referred to [14] for the details of the algorithm. Figure 4 gives the independent paths of the graph in Figure 1.
4. A CASE STUDY

The testing tool HyperTest has been used in a case study for testing a Claris on-line help system. Claris Help systems are on-line documentation systems designed to help users utilise various facilities provided by both MS-Window and Macintosh platforms. It represents general trends in the packaging of software: slim-line manuals and less hard copy support and the move towards on-line solutions. As documentation gets smaller, Help systems will get bigger and the resources needed to localise these systems will increase.

The case study takes a database Help system, which is originally developed in English and then localised into several other languages. Table 2 gives the results of the structural testing of the localised systems in several languages. The number of errors indicates inconsistency of links (jumps / pop-ups) with respect to the original Help system in English. All these systems have passed manual testing in Claris before tested on HyperTest. The total number of jumps / pop-ups in each system is around 720. By application of the testing tool, the structural testing of the four localised versions of the system is completed within a couple of hours, and the testing reports are generated automatically. The preliminary results show that the testing tool can significantly improve the efficiency and accuracy of testing.
Table 2. Results of a case study

<table>
<thead>
<tr>
<th>Language</th>
<th>Number of errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>French</td>
<td>15</td>
</tr>
<tr>
<td>Spanish</td>
<td>7</td>
</tr>
<tr>
<td>German</td>
<td>22</td>
</tr>
<tr>
<td>Norwegian</td>
<td>6</td>
</tr>
</tbody>
</table>

5. CONCLUSION

In this paper, we have analysed the requirements of testing hypertext applications. Adequacy criteria are proposed to meet the requirements. A testing tool which can automatically generate test suites to meet these requirements is described. It also supports the regression testing of localised hypertext applications.

5.1. Related Work

The use of independent paths in testing hypertext applications was inspired by McCabe's work on using the cyclomatic number as a measurement of program complexity [12], which was then extended as a test adequacy criterion for testing programs [13]. However, McCabe's method to identify testing paths is not applicable because of the differences in the structures of programs and hypertexts. In a flow graph model of program structure, there are nodes which are clearly exit nodes where program terminates, whereas in a hypertext such nodes may not be obvious. It is normal that the user of hypertext can finish the search wherever he wants.

Another important difference between programs and hypertext applications is that every path in a hypertext application is feasible in the sense that users can always walk through the path by following the arcs, while in a program a path may not be feasible, i.e. there may not be input data which cause the execution along the path.

McCabe's cyclomatic criterion for testing programs is a rather weak criterion because it requires little about the exercise of loops in a program. While testing loops many times is essential for program testing, it is not needed in the test of hypertext applications. Therefore, we believe that the independent path coverage criterion for hypertext applications is fairly strong.

5.2. Further Work

The static analyser in HyperTest, which supports the regression testing of localised hypertext applications, is developed under the assumption that the structure of the network is unchanged during localisation. In case the network is modified, either by changing the links or by adding or deleting nodes, a subset of paths of the test suite should be modified accordingly. The generation of a new test suite and testing the whole hypertext after some small
modifications is unnecessary. Hence, the generation and execution of a subset of test suite which only test the part of the graph modified is desirable. We believe that the method proposed in this paper can be easily extended into incremental algorithms to meet this requirement.

ACKNOWLEDGEMENT

The authors would like to thank Z. M. Zhao of Institute of Education Technology, The Open University, and Niamh McHugh of Claris Ireland for their valuable comments.

REFERENCE