Abstract

Architecture–based reliability analysis is necessary for a software application that is developed using the component–based software development paradigm. Prevalent architecture–based analysis techniques represent the application architecture by a Markov process, which may be adequate in the context of an application with a general–purpose architecture. The Markov process, however, is not adequate to represent the architecture of an application which follows a specific architecture style, and hence prevalent techniques cannot be used to analyze the reliability of such an application. In this paper we develop a reliability analysis methodology for an application which follows one such architecture style, namely, the pipe and filter architecture style. We consider two variants of the topological organization of the pipes and filters in an application. In the first variant the pipes and filters are organized into a linear topology, whereas the second variant consists of a linear topology with a feedback loop. The objective of the reliability analysis methodology is to develop an analytical function which expresses the overall application reliability in terms of the reliabilities of the pipes and filters, and the characteristics of the topological organization of the pipes and filters. We illustrate the potential of the methodology to obtain a reliability estimate as well as to facilitate sensitivity analysis for a Document Analysis and Understanding Application which follows the pipe and filter architecture style. To the best of our knowledge, this is the first comprehensive effort to combine two mature, yet independent research areas, namely, software reliability analysis and software architecture.

1 Introduction and motivation

Software architecture is receiving increasing appreciation as a critical design level for software systems, as they continue to grow in size and complexity. Software architecture is concerned with many aspects such as high–level component organization, protocols for communication, synchronization and data access, global control structures, performance, assignment of functionality to design elements, and the selection amongst design alternatives [9]. The components comprising an architecture could include objects, clients and servers, databases, filters and layers. These components could interact using several mechanisms such as procedure calls, message sending, shared variable access, client–server protocols, and asynchronous event multicast. A general purpose software architecture can be defined in terms of its components, component interactions, constraints, and control structure. Over the years, however, some commonly occurring patterns of the structural organization of components and connectors have been identified, and these patterns are referred to as architecture styles. An architecture style defines a family of software systems in terms of a pattern of software organization [9]. A definition of an architectural style consists of the design vocabulary or the types of components and connectors, allowable structural patterns, underlying computation pattern, essential invariants of the style, common examples, advantages and disadvantages and common specializations. Instead of a general purpose architecture, a specific architecture style may be better suited for a particular type of applications1. Examples of software architecture styles include the pipe and filter style, event–based, implicit invocation style, etc. [9]. These styles differ from one another in the way control and data are handled.

Software architecture represents the design decisions that are made in the early phases of a system and these decisions are usually difficult to change or reverse. These architectural choices have a profound influence on the non functional attributes that can be supported by a system. Software architecture analysis can be used to assess the degree to which a given software architecture supports im-

---

1The terms application, software application, software system, and system are used interchangeably in this paper
portant quality attributes such as maintainability, reliability, reusability and performance. Many different techniques have appeared for the analysis of software architectures for their non functional attributes such as performance [13, 19], maintainability [7], evolution and reusability [16], flexibility [15] and modifiability [8]. Techniques to tradeoff multiple non functional attributes have also been developed [14].

In accordance with the growing importance of analyzing the architecture of a software application for its non functional attributes, architecture–based software reliability analysis techniques have been the focus of a large number of research efforts in the past few years [11, 21, 12]. Prevalent architecture–based reliability analysis techniques rely on the use of state–space models and obtain an estimate of the application reliability using either analytical or analytical/numerical methods. State–space techniques assume that the application architecture can be represented using a first–order Markov process. The Markovian assumption underlying the state–space techniques is easily violated by an application which follows a specific architecture style [9]. Hence prevalent techniques cannot be used to analyze the reliability of such an application.

Architecture styles are widely expected to be used in the context of applications whose failures will have serious consequences including economic damage and catastrophic loss of life. For example, the pipe and filter architecture style is typically observed in the case of signal processing and remote sensing applications which may be aboard space missions. Signal processing applications could also play a vital role in medical imaging applications. Similarly, the implicit invocation or the event–based architecture style is the cornerstone of the open API environment being defined by Sun Microsystems in order to facilitate rapid creation of Next Generation Network (NGN) applications [10]. Thus, a sound, scientific methodology to assess the reliability of an application which follows a specific architecture style is essential.

In this paper we develop a reliability analysis methodology for a software application which follows the pipe and filter architecture style. We consider two variants of the topological organization of the pipes and filters in an application. In the first variant, the pipes and filters are organized into a linear topology, and in the second variant the linear topology includes a feedback loop. The primary objective of the analysis methodology is to obtain an analytical function which expresses application reliability in terms of the reliabilities of the individual pipes and filters and the topological characteristics of the organization of the pipes and filters in an application. The analytical function enables efficient computation of application reliability, and hence facilitates sensitivity analysis. We illustrate the potential of the methodology for the purpose of obtaining an estimate of the application reliability as well as for sensitivity analysis using a Document Analysis and Understanding Application. While sporadic efforts to assess the reliability an application which follows the pipe and filter style have appeared in the literature [3, 18], these efforts can consider a very limited subset of the characteristics of the pipe and filter architecture style since they try to map the organization of the pipes and filters to a well–known, standard modeling paradigm such as a Markov model or a reliability block diagram. To the best of our knowledge, this is the first effort aimed at providing a comprehensive reliability analysis methodology for an application which follows the pipe and filter architecture style.

The rest of the paper is organized as follows: Section 2 provides a brief overview of the pipe and filter architecture style. In this section, we also discuss why the Markov models used by prevalent architecture–based techniques are unsuitable to characterize the architecture of an application based on this style. Section 3 presents the reliability analysis methodology. Section 4 illustrates the reliability analysis methodology presented in Section 3 using a Document Analysis and Understanding Application. Section 5 presents conclusions and directions for future research.

2 Overview of Pipe and Filter architecture style

The reliability analysis methodology developed in this paper focuses on the pipe and filter architecture style, and this style is reviewed in this section. We also discuss why the prevalent architecture–based analysis techniques cannot be used to assess the reliability of an application which follows this style.

In the case of an application which follows the pipe and filter architecture style, each component has a set of inputs and outputs. A component reads a stream of data on its input, and produces a stream of data on its outputs. Input is transformed both locally and incrementally so that output begins before the entire input stream is consumed. Components are termed filters; connectors serve as conduits for the information streams and are termed pipes. The pipes may be bounded or unbounded. The pipes and the filters may be connected to form a generic topology as shown in Figure 1. The main characteristics of this style include the condition that filters must be independent entities, and they need not know the identities of the upstream and the downstream filters. They may specify the input format and guarantee what appears on the output, but they may not know which components appear at the ends of those pipes. The correctness of the output of a network of pipes and filters should not depend on the order in which the filters perform their incremental processing.

Existing architecture–based analysis techniques represent the architecture of a software application using a
Markov chain, where a state in the chain corresponds to the execution of a single component of the application. In the Markov model, the component that is executed next depends on the component that is presently executing. However, in the pipe and filter style, filters are assumed to be independent entities, where each filter is not aware of the identities of the upstream and downstream filters. Thus, the prevalent architecture-based techniques cannot take into consideration the constraints of the pipe and filter architectural style.

3 Reliability analysis methodology

In this section we present the reliability analysis methodology for an application which follows the pipe and filter architectural style. Towards this end, we first describe the parameters for the pipe and filter architecture and the notation used to represent these parameters.

We consider a linear topology of $n$ filters, labeled sequentially from 1 through $n$ starting at the left. The filters are connected by pipes, and each pipe is labeled $(a, b)$, where $a$ is the label of the filter that feeds data into the pipe, and $b$ is the label of the filter that consumes data from the pipe. The linear topology has $n-1$ pipes, and the pipes are labeled $(a, a+1)$, where $a$ ranges from 1 through $n-1$.

We consider two cases of the linear topology. In the first case, termed as “linear topology without feedback”, each element of input data is processed sequentially starting from the first filter all the way to the $n^{th}$ filter. Figure 2 shows pipes and filters organized into a linear topology without feedback. The second case includes a feedback loop in the topology, and is termed as “linear topology with feedback”. We assume that the feedback loop covers filters $n_1$ through $n_2$ as shown in Figure 3. Thus, each element of input data is processed by filters 1 through $n_1-1$ and by filters $n_2+1$ through $n$ just once, and may be processed multiple times by filters $n_1$ through $n_2$. A feedback loop may be incorporated into the architecture for one of two purposes. The first purpose is to improve the quality of the output and the second purpose is to improve the reliability of the processing. In both these cases, additional checks are employed to assess the intermediate results obtained after the processing by filter $n_2$ is completed. This check first determines whether a failure has occurred after processing is completed by filter $n_2$.

If no failure has occurred, and if the feedback loop has been incorporated to improve quality then a further analysis is conducted to assess whether the output from filter $n_2$ satisfies the quality considerations. If it is determined that the output does not satisfy the desired quality concerns, then the output is discarded, and the data element is subject to another round of processing by filters $n_1$ through $n_2$. This process is repeated until either the quality of the output obtained from filter $n_2$ is satisfactory or a certain maximum number of iterations denoted $m$ are completed. We let $q_r$ denote the probability that the output obtained from filter $n_2$ in the $r^{th}$ iteration satisfies the quality considerations. In this case, each time the data element is processed by filters $n_1$ through $n_2$ in order to improve the quality, there is a risk that the filter may fail. However, by the virtue of the fact that the data element was processed correctly in the previous rounds, the probability of failure decreases for each subsequent round of processing. Thus, the probability of processing the element correctly for filters $n_1$ through $n_2$ is an increasing function of the number of data processing iteration.

On the other hand, if a feedback loop has been included to improve the reliability, and if no failure is deemed to have occur after the processing of filter $n_2$, then the output of filter $n_2$ is fed into the downstream filter $n_2+1$. However, if a failure is determined to have occur upon the processing filter $n_2$, then the data is subjected to another round of processing by filters $n_1$ through $n_2$. This process is repeated until the either the output of filter $n_2$ is determined to be correct or a certain maximum number of the processing iterations denoted $m$ are reached. Due to the benefit of the experience gained from previous rounds of processing, the probability of failure of each one of the filters from $n_1$ through $n_2$ decreases with each subsequent iteration. Thus, similar to the previous case, the probability of providing correct output for filters $n_1$ through $n_2$ increases as the number of iteration increases.
Each filter receives one element of input data at a time and it processes that element. The input to the first filter is provided by an external source, while the inputs to filters 2 through n are provided by the outputs of the upstream filters 1 through n – 1 respectively. The input/output relationships among the filters described above holds even in the case of the topology with a feedback loop. In this case, if additional rounds of processing by filters n1 through n2 is deemed to be necessary, then the input to filter n1 for these additional rounds is received from filter n1 – 1 as in the first round of processing and not from filter n2. Thus, the feedback loop is a control loop and not a data loop. The processing of each element of input data by a filter yields three possibilities, it either produces a correct result, it produces an incorrect result, or it does not produce any output. Which one of these three possibilities is considered as a failure of the filter depends on the policy that is employed. We consider two policies, namely, conservative and opportunistic. In the conservative policy, whenever a filter produces incorrect result or no result, it is considered to have failed. On the other hand, in the opportunistic policy, a filter is considered to have failed only when it produces no output. If a filter produces an incorrect output, it is passed on to the subsequent downstream filter for further processing. Thus, in this policy, the downstream filters have the opportunity to correct the error committed by an upstream filter and provide correct output despite the fact that one or more of the filters may have produced an incorrect result. Due to this additional opportunity provided to remedy the failure(s), this policy is termed opportunistic. Assuming that the input received from external sources is always correct, in the conservative policy, the inputs received by each filter are always correct. On the other hand, in the opportunistic policy, the inputs received by each one of the filters except for the first one, can fall into two categories, namely, correct and incorrect.

We consider four scenarios based on the combination of the policies and the topologies described above. The simplest scenario is that of a linear topology without feedback using the conservative policy. In order to improve the reliability in the case of linear topology without feedback, we also consider opportunistic policy. In the case of linear topology with feedback, we employ only the conservative policy, since in this case the probability of producing correct output increases with each subsequent iteration through the feedback loop.

For each filter, we use the following notation to represent the conditional probabilities, inputs and total probabilities:

- \( p_j(C) \) – Probability that filter \( j \) produces correct output.
- \( p_j(I) \) – Probability that filter \( j \) produces incorrect output.
- \( p_j(N) \) – Probability that filter \( j \) produces no output.
- \( p_j(C|C) \) – Probability that filter \( j \) produces correct output upon receiving correct input.
- \( p_j(I|C) \) – Probability that filter \( j \) produces incorrect output upon receiving correct input.
- \( p_j(I|I) \) – Probability that filter \( j \) produces incorrect output upon receiving incorrect input.
- \( p_j(N|C) \) – Probability that filter \( j \) produces no output upon receiving correct input.
- \( p_j(N|I) \) – Probability that filter \( j \) produces no output upon receiving incorrect input.
- \( I_j(C) \) – Probability that filter \( j \) receives correct input.
- \( I_j(I) \) – Probability that filter \( j \) receives incorrect input.

Using the theorem of total probability [17], for each filter \( j \), the probability of correct output \( p_j(C) \), incorrect output
of input data correctly.

Case I: Topology without feedback, conservative policy

In this case, since the incorrect output of each filter is regarded as a failure and is not forwarded to the downstream filter for further processing, for each filter including the first one \( I_1(I) = 0 \), and \( I_1(C) = 1 \). Thus, from Equations (1), (2), and (3) \( p_j(C) = p_j(C|C) \), \( p_j(I) = p_j(I|C) \) and \( p_j(N) = p_j(N|C) \). The probability of processing each element of input data correctly or the reliability is given by:

\[
R = \left( \prod_{i=1}^{n} p_j(C) \right) \left( \prod_{i=1}^{n-1} p_i, i+1 \right)
\]  

Case II: Topology without feedback, opportunistic policy

In this case, filter \( j \) \((j > 1)\) receives both incorrect and correct inputs, when the outputs produced by its upstream filter \( j - 1 \) are incorrect and correct respectively. Thus, based on the probabilities of correct and incorrect output of filter \( j - 1 \), namely, \( p_{j-1}(C) \) and \( p_{j-1}(I) \), \( I_j(C) \) and \( I_j(I) \) can be obtained using the following expressions:

\[
I_j(C) = \frac{p_{j-1}(C)}{p_{j-1}(C) + p_{j-1}(I)} \\
I_j(I) = \frac{p_{j-1}(I)}{p_{j-1}(I) + p_{j-1}(C)}
\]  

The normalization in Equations (6) and (7) is necessary since the output of filter \( j - 1 \) can be of three types, only two of which are fed into filter \( j \) for further processing. When filter \( j - 1 \) does not produce any output, no input is fed into the downstream filter \( j \) and the application is considered to have failed.

Substituting the expressions for \( I_j(C) \) and \( I_j(I) \) from Equations (6) and (7), Equations (1), (2) and (3) for filter \( j \) can be written as:

\[
p_j(C) = \frac{p_{j-1}(C)}{p_{j-1}(C) + p_{j-1}(I)} p_j(C|C) + \frac{p_{j-1}(I)}{p_{j-1}(C) + p_{j-1}(I)} p_j(C|I)
\]  

\[
p_j(I) = \frac{p_{j-1}(C)}{p_{j-1}(C) + p_{j-1}(I)} p_j(I|C) + \frac{p_{j-1}(I)}{p_{j-1}(C) + p_{j-1}(I)} p_j(I|I)
\]  

\[
p_j(N) = \frac{p_{j-1}(C)}{p_{j-1}(C) + p_{j-1}(I)} p_j(N|C) + \frac{p_{j-1}(I)}{p_{j-1}(C) + p_{j-1}(I)} p_j(N|I)
\]
Equations (9), (10), and (11) hold for filters 2 through \( n \).

For the first filter, assuming that the input received from the external source is always correct, \( p_1(C) \) is given by \( p_1(C)C \) and \( p_1(I) \) is given by \( p_1(I)C \). Then, the reliability or the probability of processing one element of data correctly is given by:

\[
R = \left( \prod_{i=1}^{n-1} (p_i(C) + p_i(I)) \right) p_n(C) \left( \prod_{i=1}^{n-1} p_{i+1} \right) \quad (11)
\]

Equation (11) indicates that in the case of filters 1 through \( n - 1 \), the possibilities of producing both correct and incorrect output are regarded as success. In the case of filter \( n \), however, only the scenario of producing correct output is regarded as success, while the scenarios of producing incorrect output and no output are regarded as failure.

**Case III: Topology with feedback for quality improvement, conservative policy**

In this case, each element of input data is processed by filters 1 through \( n_1 - 1 \) and filters \( n_2 + 1 \) through \( n \) exactly once. However, the data may be processed multiple times by filters \( n_1 \) through \( n_2 \) due to the presence of a feedback loop. In order to obtain an expression for the overall application reliability, we first need to obtain an expression for the average reliability for multiple processing iterations by filters \( n_1 \) through \( n_2 \). In the first round of processing, the probability that filters \( n_1 \) through \( n_2 \) process an element of input data without incurring a failure is given by \( \prod_{j=n_1}^{n_2} p_j(C) \prod_{j=n_1}^{n_2-1} p_{j+1} \). With probability \( q_1 \) the output produced by the cascade of filters \( n_1 \) through \( n_2 \) is deemed to be of acceptable quality. Thus, with probability \( (1 - q_1) \) the output produced is of unacceptable quality requiring a second round of processing. In the second round of processing, the probability that processing through the cascade of filters \( n_1 \) through \( n_2 \) does not encounter a failure is given by \( \prod_{j=n_1}^{n_2} p_j(C) \prod_{j=n_1}^{n_2-1} p_{j+1} \). With probability \( q_2 \) this output is deemed to be of acceptable quality requiring no further action, and with probability \( (1 - q_2) \) a third round of processing is necessary. Using the theorem of total probability [17], the probability that an output of acceptable quality is obtained at the \( l^{th} \) iteration is given by

\[
(W(C))^{l+1} \prod_{i=1}^{l} W_p(l) \]

where \( W(C) = \prod_{j=n_1}^{n_2} p_j(C) \), and \( W_p(l) = \prod_{j=n_1}^{n_2-1} p_{j+1} \). The extreme case, namely, \( l = m \) merits special consideration. In this case, the probability of obtaining acceptable quality output is \( (1 - q_1)(1 - q_2) \ldots (1 - q_{m-1}) \). This expression implicitly assumes that \( q_m = 1 \). In other words, if an output of acceptable quality cannot be obtained after \( m - 1 \) iterations through the cascade of filters \( n_1 \) through \( n_2 \), then the output produced in the \( m^{th} \) iteration is considered acceptable as long as no failure is encountered. The average

![Figure 4. \( p_j(V, C) \) as a function of \( p_j(C) \) and \( d \)](image-url)
reliability of processing by the cascade of filters \( n_1 \) through \( n_2 \), denoted \( W_{III} \) is given by:

\[
W_{III} = W(C)W_pq_1 + \sum_{i=1}^{m-1} \prod_{l=2}^{i+1} (1 - q_l)q_l \tag{12}
\]

\[
(W(C))^{1+\sum_{i=2}^{n} i} W_p^l + \prod_{l=1}^{n-1} (1 - q_i)W(C)^{1+\sum_{i=2}^{n} i} W_p^m
\]

The overall reliability of the application is then given by:

\[
R = (\prod_{i=1}^{n_1} (p_i(C) * p_{i+1}))(W_{III})(\prod_{i=n_2+1}^{n} (p_i(C) * p_{i-1,i})) \tag{13}
\]

Since the conservative policy is employed, \( p_j(C) \) is the same as \( p_j(C|C) \) for all the filters.

**Case IV: Topology with feedback for reliability enhancement, conservative policy**

Similar to the case where the feedback is employed for quality enhancement, in this case, each element of input data is processed by filters \( n_1 - 1 \) and filters \( n_2 + 1 \) through \( n \) exactly once. However, the data may be processed multiple times by filters \( n_1 \) through \( n_2 \) due to the presence of a feedback loop. We obtain an expression for the expected reliability for multiple processing iterations by filters \( n_1 \) through \( n_2 \) using the following argument. In the first round of processing, if a failure is determined to have occur after processing by filter \( n_2 \) is complete, then a second round of processing is necessary. If the second round of processing produces also results in a failure, then a third round of processing becomes necessary. This can continue until either a correct output is produced or until all \( m \) iterations are exhausted. Using the theorem of total probability, the probability that the \( lth \) iteration will be required is given by \( \prod_{i=1}^{l-1} (1 - W(C))^{1+\sum_{j=2}^{n} l} W_p \), for \( 2 \leq l < m \). \( W(C) \) is given by \( \prod_{i=n_1}^{n_2} p_i(C) \), and \( W_p \) is given by \( \prod_{j=n_2}^{n} p_j, j+1 \). The probability that the \( lth \) iteration produces correct output is given by \( W(C)\frac{1}{W_p} \). The probability that the first iteration \( l = 1 \) produces correct output is given by \( W(C)W_p \). The average reliability of the filters in the feedback loop denoted by \( W_{IV} \) is given by:

\[
W_{IV} = W(C)W_p + (1 - W(C)W_p)(W(C))\frac{1}{W_p}(1 + W(C)) \tag{14}
\]

\[
+(1 - W(C)W_p)(\sum_{l=2}^{m} \prod_{i=2}^{l-1} (1 - W(C))\frac{1}{W_p}))
\]

\[
W(C)\frac{1}{W_p}
\]

The overall reliability of the application is then given by:

\[
R = (\prod_{i=1}^{n_1} (p_i(C) * p_{i+1}))(W_{IV})(\prod_{i=n_2+1}^{n} (p_i(C) * p_{i-1,i})) \tag{15}
\]

Since the conservative policy is employed, \( p_j(C) \) is the same as \( p_j(C|C) \) for all the filters.

**4 Case study: Document Analysis and Understanding System**

To illustrate the application of the proposed methodology for reliability analysis of a software application which follows the pipe and filter style, we use the architecture of a Document Analysis and Understanding System.

Document understanding refers to the process of converting paper material such as books, magazines, journals, etc. into a searchable electronic form with information that is meaningful to both human beings and machines [20]. Sometimes this conversion process is referred to as “remastering”. Typically, the system input is document pages in a raster format (TIFF or BMP) and the output is searchable Portable Document Format (PDF) or eXtensible Markup Language (XML) documents. Document understanding systems often remaster documents for use in the digital libraries of Web communities, such as the cognitive science community [1].

Currently, there are several commercial applications that provide various document understanding services, including WISDOM++ 1.2 [5, 6] Abbyy FineReader [2], and Abode Acrobat Capture [4]. Most existing solutions are considered applications or programs, rather than systems. Applications are useful for end–users because they provide graphical user interfaces through which the user interacts with the system, accepts or rejects results, or resubmits the document for processing under different configurations. We refer to these solutions as standalone applications; they are unsuited for high–volume data processing because they require extensive human intervention. To process such a massive amount of data in a reasonable amount of time, we need an automated system that runs continuously, with minimal human intervention, and operates on multiple documents at the same time.

From a software perspective, the architecture is based on component-based software engineering principles. Components perform the document understanding functions, such as optical character recognition (OCR), layout analysis, and logical structure analysis. Each component is self–contained and fairly independent, and provides well–defined services and functions. We wrap each component such that the wrapper handles inputs, outputs, and errors,
and provides the execution context. The architecture is built using a basic software framework principle: “Don’t call us, we will call you”. Components have no explicit knowledge of each other, whether they are on the same worker machine or not. Processes run in parallel over a distributed workstation cluster. Figure 5 illustrates the software architecture of the application.

The system is composed of multiple components and scripts that run in parallel with monitoring components, watchdogs, loggers, etc. For simplicity, we extract the most critical portion of the application to apply our analysis techniques. These are the pipes and filters responsible for the analysis of the documents. The following figure depicts a simplified diagram for the pipeline that we will use for the purpose of illustration.

Note that for this analysis, the pipes are control pipes; that is, they carry a description of what information will be processed but not the data itself. For this type of applications, the data itself is large (in Megabytes) and hence a storage server is used to store the data. The location of the data to be processed is carried in the task description which goes into the pipe.

From an implementation point of view, the filters are executable components (as .exe or Java classes) wrapped in shell scripts (for data manipulation and component monitoring).

The following figure depicts the same architecture when a feedback loop is added. In this simplified view of the architecture, the most critical component is the Content Analyzer / Converter filter which performs most of the analysis. To improve the reliability of the converter filter, a feedback loop is added whose function is to maximize the possibility of getting a higher quality output. The feedback loop works as follows: the output from the PDF verifier is used to determine whether the output has reached acceptable quality or not. If yes, the output is fed to the writer filter. If not, the feedback loop is used to reinvoke the converter to redo the analysis. The analysis is performed in every feedback turn with different parameters and settings. For this system, a set of 5 different types of analyzers are used; some of which use the same technology with different parameter values (for example changing the exposure in the input page) and others use different analysis technology (for example using polygonal region analysis or quadratic).

The conditional probabilities of the filters are shown in Table 1. The rationale behind this initial choice of these probabilities is as follows. The Tape Loader uploads the data from the archive tapes into the system. We observed only one type of errors in this component, the component may not write the data correctly to the system disks \( p(N|C) \). It will never produce correct output given incorrect input, hence \( p(C|I) \) is set to zero. The Writer filter component has proven to be sufficiently reliable so we set its \( p(C|C) \) to one. The filter reads output data and groups it into manageable writable disks. The Input Tiff filter component reads the input and checks for its quality. The component may decide that the input is correct while it is not and hence there is a likelihood value set to \( p(C|I) \) and there is also a likelihood that it will produce incorrect output given that the input was correct \( p(I|C) \) however, the second case is less likely and most of the errors observed from that component belong to the first category. The PDF Verifier is another reliable component that is based on a stable commercial component. This component was carefully selected to be highly reliable since based on its output, a decision is made whether the output is acceptable or not. The Content Analyzer is the most critical and error prone component. It is a complex component that encompasses sophisticated image processing and analysis algorithms. It is the most important component in the system, at the same time it is the one that is most likely to fail. Therefore, a feedback loop is used to decrease the likelihood of failures of this component. For the feedback loop, the values of \( q_1, q_2, q_3, q_4 \) and \( q_5 \) are set to 0.96, 0.97, 0.98, 0.99 and 1.00 respectively. Since the number of iterations is 5, \( q_5 \) is set to 1.0. The value of \( d \) is set to 1.00, and the value of \( m \) is set to 5.

With these values of the parameters, the application reliability for the four scenarios was computed using the reliability expressions derived in Section 3, and these values are summarized in Table 2. Table 2 indicates that the application reliability is the lowest for linear topology without feedback, and is the highest for linear topology with feedback when the feedback is employed for either quality or reliability improvement. Intuitively this is expected, however, the expressions derived in this paper allow us to quantitatively assess the reliability improvement afforded by each strategy. When the feedback is employed for quality improvement, an improvement in the reliability is also enabled as a fringe benefit.

Next, we illustrate the potential of the reliability expressions developed in this paper to analyze the sensitivity of the application reliability to \( p(C|C) \)'s of each filter. Towards this end, we varied \( p(C|C) \) of each filter one at a time, such that \( p(C|C) + p(I|C) = 1.0 \), and setting \( p(N|C) \) to be 0.0. When \( p(C|C) \) of a given filter were varied, the remaining parameters of that filter, as well as the parameters of all the other filters were held at the values shown.

<table>
<thead>
<tr>
<th>Case</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without feedback, conservative</td>
<td>0.8402</td>
</tr>
<tr>
<td>Without feedback, opportunistic</td>
<td>0.8519</td>
</tr>
<tr>
<td>With feedback (quality improvement)</td>
<td>0.9020</td>
</tr>
<tr>
<td>With feedback (reliability improvement)</td>
<td>0.9020</td>
</tr>
</tbody>
</table>
Figure 5. Software Architecture of the Document Analysis and Understanding System

Figure 6. Pipe/Filter architecture of Document Understanding and Analysis System (without feedback)

Figure 7. Pipe/Filter architecture of Document Understanding and Analysis System (with feedback)
Table 1. Conditional probabilities for the filters

| Filter          | \(p(C|C)\) | \(p(I|C)\) | \(p(N|C)\) | \(p(C|I)\) | \(p(I|I)\) | \(p(N|I)\) |
|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|
| Tape Loaders    | 0.95        | 0.00        | 0.05        | 0.00        | 0.00        | 0.00        |
| Input Tiff      | 0.95        | 0.025       | 0.025       | 0.5         | 0.492       | 0.008       |
| Content Analyzer| 0.95        | 0.03        | 0.02        | 0.5         | 0.492       | 0.008       |
| PDF Verifier    | 0.98        | 0.005       | 0.015       | 0.00        | 1.00        | 0.00        |
| Writer Filter   | 1.00        | 0.00        | 0.00        | 0.00        | 1.00        | 0.00        |

in Table 1. \(p(C|C)\)'s were varied between 0.60 to 1.00 in steps of 0.05, and the application reliability was computed for the linear topology without feedback using both the conservative and the opportunistic policies. Figures 8, 9, 10, 11, and 12 show the variation in application reliability for variations in \(p(C|C)\)'s for the Tape Loaders filter, Input Tiff filter, Content Analyzer filter, PDF Verifier filter, and Writer filter. As expected, in all the five figures the opportunistic policy provides better reliability than the conservative policy. However, for lower values of \(p(C|C)\), the difference in the reliability estimate between the opportunistic and conservative policies is higher in the case of Tape Loaders and Input Tiff filters. For the Content Analyzer, PDF Verifier and Writer filters, the difference in the reliability estimate between the opportunistic and the conservative policies is uniform over the entire range of variation of \(p(C|C)\). Referring to the topology of the pipes and filters in the Document Analysis System in Figure 6, it can be seen that the Tape Loaders and the Input Tiff filters occur before the remaining three filters. As a result, the higher probability of incorrect processing of an element of input data by these two filters, can be partly compensated by the downstream filters in the opportunistic policy. This indicates that if limited resources are available then it may be better to enhance the reliability of the downstream filters and to provide them with an ability to mask the errors committed by the upstream filters in order to improve the application reliability. The expressions developed in this paper can facilitate such tradeoffs.

Figures 10 and 11 indicate that the opportunistic policy does not provide a significant improvement in the application reliability over the conservative policy, even for lower values of \(p(C|C)\) of Content Analyzer and PDF verifier filters. Another way to improve the application reliability is to employ a feedback loop which covers the Content Analyzer and the PDF Verifier filters. Figure 13 shows the application reliability for linear topology without feedback and with feedback as a function of \(p(C|C)\) for the Content Analyzer (left figure) and PDF Verifier (right figure) filters respectively. The figures indicate that employing a feedback loop enables a significant improvement in the application reliability for lower values of \(p(C|C)\). In fact, when a feedback loop is employed, the reliability of the cascade of Content Analyzer and PDF Verifier filters approaches unity.
over the entire range of $p(C|C)$, and thus application reliability in this case is mainly governed by the reliabilities of the Tape Loaders, Input Tiff and Writer filters and is hence constant over the entire range of $p(C|C)$.

5 Conclusions and future research

In this paper we presented a reliability analysis methodology for a software application which follows the pipe and filter architecture style. We considered two variants of the pipe and filter architecture style. In the first variant the pipes and filters are organized into a linear topology, whereas the second variant consists of a linear topology with a feedback loop. The analysis methodology generates analytical functions which relate the application reliability to the reliabilities of the pipes and filters and the characteristics of the topological organization of the pipes and filters. We illustrate the potential of the methodology to obtain a reliability estimate as well as to facilitate sensitivity analysis for a Document Analysis and Understanding Application which follows the pipe and filter architecture style. To the best of our knowledge, this is the first comprehensive effort to combine two mature, yet independent research areas, namely, software reliability analysis and software architecture. Our future research consists of developing reliability analysis methodologies for other software architecture styles.

References

Figure 13. Reliability for variation in $p(C|C)$ of Content Analyzer and PDF Verifier Filters (With and without feedback)


