FS InfoCat

Comprehensive Summary

Leonard T. Erwine

Western Governors University

000356334

Comprehensive Summary

FS InfoCat is a file cataloging Windows Desktop application designed to support managing files distributed across multiple storage locations and devices. This application collects metadata and characteristic of files, storing the information in a database, which can then be searched and analyzed.

Our local church was experiencing problems managing and finding files from many storage devices, which lacked sufficient structure. As their storage requirements increased, the cost-effectiveness of file storage subscriptions decreased. Although they had written policy and governance on file storage practices, it was difficult to verify and enforce without causing disproportionate productivity degradation. The lack of compliance was mainly due to the inherent transiency of the primarily volunteer and part-time staff. There were many instances where someone had to re-create something that someone had already done before, simply because nobody knew where to find the files. They also found themselves periodically running out of storage space. The prospect of manually cleaning up the common storage areas was at least as costly as the price of increasing storage capacity. Due to the uncertainty of the actual content within the shared drives, it was impossible to assess the cost in person-hours, should they decide to go through each file to determine what can be consolidated or discarded. It is also conceivable that the pace of unmanaged data contributions could outpace the mitigation efforts. Even if they did reduce the storage footprint and reduce the cost of file storage services, that would not have solved the problem of recalling specific stored files when needed.

FS InfoCat was designed to create a database catalog of metadata from files within multiple disparate file systems, including network drives, local hard drives, and removable media. Basic information, such as file location, size, and modification dates, would be gathered and stored in the database. Additionally, the software leverages the Windows Property System to collect and store other metadata from files, such as author names, audio titles, and video frame rates. Since not all devices support the Windows Property System, the application allows people to provide relevant information for each file manually.

One of the concerns expressed by a stakeholder with this type of solution was the possibility that confidential specific files could be compromised by publishing its metadata to the database. The application initially stores information to a local database within the user’s windows profile to mitigate this risk. It is only published to the centralized database when the user explicitly opts to do so. Additionally, users can configure the software to skip specific files and folders.

To assist with the mitigation of excessive storage utilization, this software checks for redundancies. It creates a cryptographic hash code of each file that shares the exact byte count as another file. Files which have the same size and cryptographic hash are highly likely to be duplicates of each other. If deemed appropriate, the user can tell the application to compare the files byte-by-byte to determine whether they are actual duplicates. In addition to the cryptographic hashing, which only indicates possible binary data duplication, other metadata can be search and compared, such as title and author. By comparing these other properties, the user can identify files that contain the same material but with different quality standards.

The original intent was to produce an application in the first iteration that could synchronize with a centralized database. However, since the shared drives are once again close to capacity, they would like to utilize the software sooner. Therefore, the first release does not include this capability. However, in this state, it is still helpful for gathering and analyzing information locally.

I designed this software using a modified Waterfall development approach. The main reason for using this method was because there was only a single developer, and none of the stakeholders had enough free time to participate in Scrum meetings. Additionally, the capabilities required in the first release would require much more time than I could fit within a typical Agile sprint. To benefit from the disciplines of other software development methodologies, I employed test-driven development techniques and used project boards intended for Agile development. The most critical components and those that are difficult to validate using event tracing or other debugging means were validated using unit tests. To track and manage use cases, storyboards and bugs, I used a GitHub project.

This software development will culminate in a single binary deployment package that the church’s system administrator can install.

FS InfoCat operates within a portable, self-contained folder, unrestricted by any pre-determined file system path. Since the user can install and execute it from any arbitrary location, the system administrator has more options for formulating recommendations to ensure the best performance within varying environmental conditions.

The software validation process has three stages: Automated testing, scripted developer testing, and user acceptance testing. Automated unit tests are created synchronously with the development process to validate each component as I implement them and expose potential bugs. The purpose of the scripted developer testing is to ensure that it is fit for use. I will script the tests to ensure all features within the graphical user interface produce the intended result. Finally, the church’s system administrator and some audio/visual team stakeholders will participate in the user acceptance testing. The user acceptance testing will ensure that the software is designed according to its intended purpose as understood by the customer.

I used Visual Studio 2019 as the development platform. This was chosen because it provided the best option for creating an application that could take advantage of Windows operating system APIs that was able to produce much of the metadata that was needed. That way more attention could be devoted toward processing and interpreting metadata instead of trying to find ways to extract the data.

(a projected timeline including milestones, start and end dates, duration for each milestone, dependencies, and resources assigned to each task)

References