1. **Project Title**: Fault-tolerant Encryptable File System

2. **Project Background**:
   Many embedded devices use Flash storage for persistent storage. In situations where power fluctuates (batteries running out, power outages, pulling the plug), the integrity of these Flash-based file systems can be compromised, rendering the embedded device degraded or useless.

   This problem is often solved by utilizing a transaction-based file system, in which write operations are atomic. However, no such file system exists today that also provides an interface for user-defined encryption.

3. **Project Objectives**:
   Design and implement an atomic C-based embedded file system easily adaptable for projects with unique encryption needs.

4. **Project Description**:
   The solution:
   - SHALL be written in ANSI C
   - SHALL be suitable for running on an embedded system
     - SHOULD be suitable to run on low-resourced systems (CPU, RAM)
     - SHOULD offer efficient storage in file system
     - SHOULD have small file system code size
   - SHALL be optimized for Flash storage (or SSD)
     - SHOULD support both NOR and NAND Flash
   - SHALL wrap a currently maintained, mature, open-source file system
     - SHALL be transaction-based (i.e. atomic write operations)
     - SHALL make minimal modifications to the open-source file system, to enable easy upgrade file system to integrate upstream bugfixes
   - SHALL present an interface for using user-defined file encryption
   - SHALL present an interface for using user-defined filename obfuscation
   - SHOULD be adaptable for use on a hard disk drive (HDD)

   Stretch goal:
   - SHOULD implement a logical layer, to bridge storage regions that are discontiguous or on separate parts altogether

   Definitions of the above terms (“shall”, “should”) are per RFC 2119.

5. **Expectations**

   Phase 1 (Design):
   - Learn about the problem space. Understand/clarify all requirements.
   - Find a reliable, repeatable way to corrupt a Flash-based file system as a result of power loss.
     - Find a good hardware platform that will enable easy testing. Arduino? Something else?
   - Perform a trade study of candidate open-source file systems.
   - Design a solution that meets all project requirements.

- Write system-level tests for each of the requirements.
    - Generate a traceability matrix to ensure that your test cases cover all of the requirements.
    - Simulate power outages during Flash writes, and ensure that file system remains intact.
    - Identify file system testing, to ensure that that your solution has not negatively affected the technical soundness of the open source implementation in any way. (Think file system integrity/correctness.)
- Conduct a Preliminary Design Review (PDR), to include the professor and Northrop advisor.
- Implement quick/dirty prototypes for the highest-risk portions of your design. Cut corners. Verify that you have good solutions to the toughest problems you think you'll face.
- Conduct a Critical Design Review (CDR), to include the professor and Northrop advisor.

Phase 2 (Implementation):

- Implement your design.
- Perform unit testing throughout (and at the conclusion of) your implementation phase.
    - Does the open source tool you chose have any unit-level tests?
    - Implement a few different types of encryption to unit-test your wrapper.
- Conduct a Test Readiness Review (TRR), to include the professor and Northrop advisor.

Phase 3 (System Testing):

- Perform system-level tests identified during Phase 1.
    - If software changes are required, document them, fix them, then repeat the test cases.
- Generate a test report.

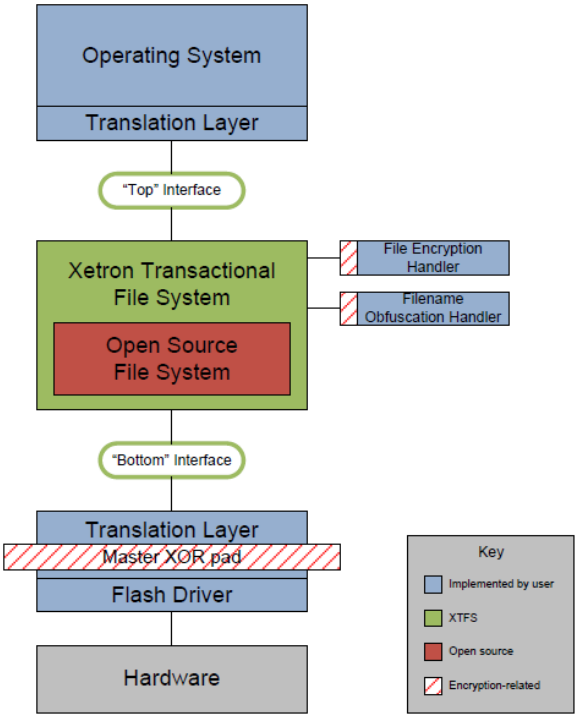
Phase 4 (Analysis):

- Analyze the strengths and weaknesses of your solution, and present your findings to your professor and Northrop advisor.
- Enumerate future work.
- *(Optional)* Test ease of merging in upstream bugfixes from open source product to your solution. If no upstream updates exist, make up your own. :)
- *(Optional)* Perform wear-level testing to quantify expected lifetime of Flash storage.

Overall Expectations:

- Use source control management (SCM) for all source code throughout the project. This includes any throw-away prototype code, as well as your implementation
    - I strongly recommend Git, as it's currently one of the industry favorites.
    - GitHub is great. Does your school have an enterprise installation?
- Apply revision control to your documentation as well.
    - Hint: You can use the same solution for documents as you did for source control, if you want. However, you don't have to.
- Deliver all generated documentation and source code repositories to your professor and Northrop advisor at the conclusion of your project.
    - Bonus if you can make your documentation and source code repositories available to your professor/advisor throughout your project.

6. **System overview**
    - The interfaces below are negotiated by Northrop and the student teams.
    - The red box below is selected by the student team (after a trade study).
    - The green box below is designed/implemented by the student team.



7. **Reference material**:

    Transactional file systems:

    - https://en.wikipedia.org/wiki/File_system#Transactional_file_systems

    List of File systems optimized for flash memory (needs to be filtered for transactional file systems):

    - https://en.wikipedia.org/wiki/List_of_file_systems#File_systems_optimized_for_flash_memory.2C_solid_state_media