IT-314 LAB 7

Program Inspection, Debugging and Static Analysis

Harsh Popatiya
202201463

Code Link : https://github.com/wxWidgets/wxWidgets/blob/master/src/common/datetime.cpp

1. How many errors are there in the program? Mention the errors you have identified.

The code does not contain outright compilation errors, but it has several **potential issues and shortcomings**:

Logical Errors

- **Month and Day Calculation Logic**:
  - In the `AddMonths` function, when adding months, if the resulting day exceeds the number of days in the new month, the code sets the day to the last valid day of that month. For example, if you add one month to January 31st, it may return February 28th or 29th, which could lead to ambiguity or unintended behavior if the user expects to stay on the last day of the month regardless of its length.

Assumption of Valid Input Dates

- **Date Range Limitations**:
  - The code's internal logic for date calculations, particularly with Julian Day Numbers (JDN), assumes that inputs will always be within a reasonable range. If a user inputs a date far in the past or future, it could lead to incorrect results. For example, the algorithm does not handle dates before 4714 BC or the potential overflow for dates beyond the maximum representable value in the underlying system.

Undefined Behavior with `DoIsHoliday`

- **Missing Implementation**:
  - The function `DoIsHoliday(dt)` is called within loops but is not defined in the provided code snippets. If this function does not exist or is incorrectly implemented, the entire process of identifying holidays will fail.

Error Handling

- **Lack of Robust Error Checking**:
    - The code does not provide adequate error handling for invalid dates, such as when `dtStart` is after `dtEnd`. If the user provides incorrect input, the code will silently return empty results without notifying the user. Implementing checks and returning meaningful error messages would improve usability significantly.

2. Which category of program inspection would you find more effective?

Formal Code Review and Static Analysis would be the most effective inspection categories for this code. Here's why:

- **Formal Code Review**:
    - This involves a detailed examination of the code by peers to catch logical errors and ensure the implementation meets the required standards. Given the complexity of date and time manipulations, having multiple sets of eyes on the logic can help uncover nuanced bugs and assumptions that one developer might miss. For instance, reviewers could focus specifically on how leap years are handled, how holidays are calculated, and whether the assumptions made in the code are valid in all scenarios.
- **Static Analysis Tools**:
    - Using tools that automatically analyze the code for potential errors (like SonarQube, ESLint, or Clang Static Analyzer) would also be beneficial. These tools can identify issues such as unreachable code, potential memory leaks, and adherence to coding standards. For example, if there's code that checks if a date is a holiday but lacks proper checks for the validity of the date, static analysis can flag this.

3. Which type of error can you not identify using the program inspection?

Dynamic Errors are a significant category that might go unnoticed during program inspection:

- **Runtime Errors**:
    - These errors occur when the code is executed rather than when it is compiled. Examples include:
        - Incorrect handling of user input (e.g., invalid dates that lead to out-of-range calculations).

        - Logical errors that only become apparent with specific data inputs, such as unexpected behavior when calculating holidays for non-standard years (like those with adjusted leap year rules).
- **Performance Issues**:
    - While static analysis can identify some inefficiencies, it cannot truly gauge how the code performs under load. For example, if a user queries a date range spanning many years, the linear iteration over each day may lead to performance bottlenecks that are only identifiable during runtime.

4. Is the program inspection technique worth applying?

Yes, program inspection techniques are absolutely worth applying, and here's why:

- **Early Detection of Bugs**:
    - Regular inspections can catch bugs before they are deployed, which is much cheaper than fixing them post-deployment. Catching logical errors in date calculations early on can save significant troubleshooting time later.
- **Improved Code Quality**:
    - Inspections encourage adherence to best practices and coding standards, which enhance the overall quality and maintainability of the code. For example, using consistent naming conventions for functions and variables can make the codebase easier to navigate.
- **Knowledge Sharing and Team Learning**:
    - Code reviews foster a culture of collaboration and learning among team members. They allow developers to share insights about date-time calculations, potential pitfalls, and alternate approaches.
- **Comprehensive Documentation**:
    - Inspections often lead to better documentation practices, which is especially important for complex functionalities like date-time handling. This documentation helps future developers understand the reasoning behind certain implementation decisions and improves the maintainability of the codebase.

Static Analysis using cppcheck:

```
Checking static_code.cpp: __WINDOWS__;wxUSE_DATETIME...
Checking static_code.cpp: wxDEBUG_LEVEL;wxUSE_DATETIME...
Checking static_code.cpp: wxHAS_STRFTIME;wxUSE_DATETIME...
Checking static_code.cpp: wxUSE_DATETIME...
Checking static_code.cpp: wxUSE_DATETIME;wxUSE_EXTENDED_RTTI...
Checking static_code.cpp: wxUSE_DATETIME;wxUSE_INTL...
static_code.cpp:94:0: style: The function 'wxStringReadValue' is never used. [unusedFunction]
template<> void wxStringReadValue(const wxString &s , wxDateTime &data )
^
static_code.cpp:99:0: style: The function 'wxStringWriteValue' is never used. [unusedFunction]
template<> void wxStringWriteValue(wxString &s , const wxDateTime &data )
^
static_code.cpp:123:0: style: The function 'OnInit' is never used. [unusedFunction]
    virtual bool OnInit() override
^
static_code.cpp:130:0: style: The function 'OnExit' is never used. [unusedFunction]
    virtual void OnExit() override
^
static_code.cpp:2426:0: style: The function 'wxPrevMonth' is never used. [unusedFunction]
WXDLLIMPEXP_BASE void wxPrevMonth(wxDateTime::Month& m)
^
static_code.cpp:2434:0: style: The function 'wxNextWDay' is never used. [unusedFunction]
WXDLLIMPEXP_BASE void wxNextWDay(wxDateTime::WeekDay& wd)
^
static_code.cpp:2442:0: style: The function 'wxPrevWDay' is never used. [unusedFunction]
WXDLLIMPEXP_BASE void wxPrevWDay(wxDateTime::WeekDay& wd)
^
nofile:0:0: information: Active checkers: 161/592 (use --checkers-report=<filename> to see details) [checkersReport]

om@om-VirtualBox:~/Desktop/SE_Code_Analysis$
```