

A brief contrast in code complexity

Why we need to refactor

David Annetts, Aegis Geophysics

February 21, 2026

Background

- The modernisation process has been underway since mid-April 2025
 - Panel A plots lines of code as δ from the previous commit
 - Panel B plots the total number of lines of Julia code
 - The final release of `Leroi` has about 12000 loc
 - This pattern is typical of refactoring where a code base is refined through addition, testing, then deletion of superfluous code

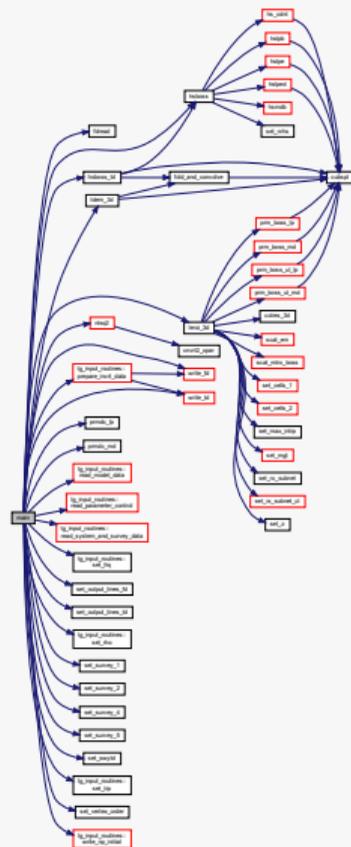


Why modernise in the first place?

- Extensive use of F77 idioms in P223 codes (Raiche et al.2007)limits maintenance & evolution
 - Employing subroutines as library calls does not evolve or develop code
 - Exacerbates technical debt
 - Limits code use as pedagogical tool
- Modules ...
 - Modules are a great idea and assist development of large complex code bases
 - Partition the problem into smaller, manageable components
 - Unfortunately, modules were poorly implemented in all P223 codes
 - Implementation like a COMMON block
 - Lots of global variables \implies potential for side effects

Why modernise in the first place?

- First and foremost, the codes are difficult to maintain
 - Codes are complex
 - Complex code requires more mental effort from developers to understand its logic and flow
 - This increased mental burden makes it easier to overlook edge cases or introduce unintended consequences when making modifications (Tashtoush et al.2023)
- Image on right shows Leroi flow structure. Nodes in red lead to more subroutines.



We know that the P223 codes are complex.
This presentation attempts to quantify how complex they are.

Code complexity

- Cognitive complexity (Gao et al.2025)
 - A newer metric that assesses the mental effort required for a human to understand the code. It specifically penalizes structures that are difficult to read, such as deep nesting, to encourage more understandable code
- Halstead complexity (Halstead1977)
 - Based on the number of unique operators and operands in the code, providing insights into the program's volume and the potential effort required to understand and maintain it
- Cyclotomic complexity (McCabe1976)
 - Measures the number of linearly independent paths through a program's source code (e.g., from if statements, for loops, switch cases). A higher score indicates a greater number of execution paths and a higher probability of errors.
- We will use Cyclotomic Complexity and note that Gao et al.2025) recommended a holistic approach using multiple metrics

- `cloc`: counts the number of lines of code
- `doxygen`: used to produce program call diagrams
- `CodeComplexity.jl`: used to compute the cyclotomic complexity of Julia code
- `npm`: used to compute the cyclotomic complexity of Fortran code

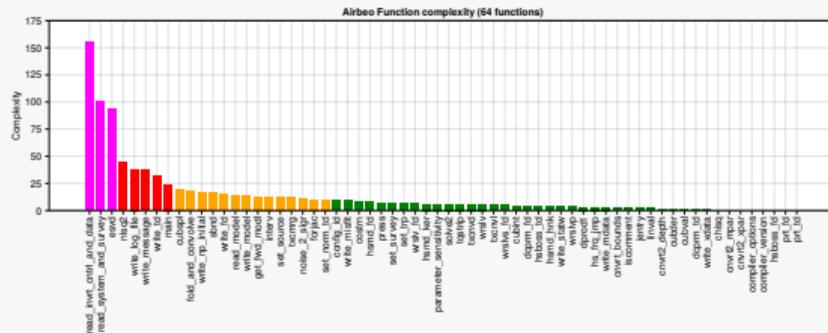
Cyclotomic complexity

- Count the branches in a routine
- Used by Wallace et al.1996) as a foundation for structured testing
- McCabe (McCabe1976) suggests categorisation on the right

CC Score	Interpretation	Risk
1 - 10	Simple procedures	Low
11 - 20	More complex procedures	Moderate
21 - 50	Complex	High
> 51	Untestable, unmaintainable	Very high

Low CC scores do not mean that code is bug-free
Low CC scores suggest that code is easier to understand and debug.
NIST recommend CC < 10; NASA recommends CC < 15

- Airbeo is designed to invert airborne EM surveys for an n -layered earth
- The program contains 64 functions over 4429 lines in a single file
 - About 69% of the code is in simple functions
 - About 4.7% code in unmaintainable functions
- This is the least complex of any of the P223 codes

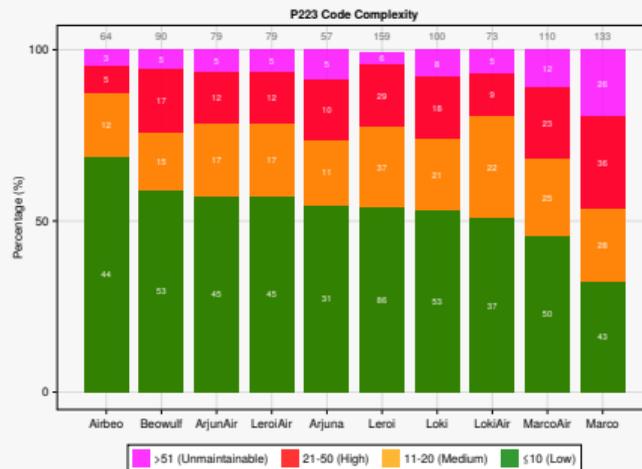


Aegis Geophysics, 2020-03-03

Complexity	Total
Simple	44
Moderate	12
High	5
Unmaintainable	3

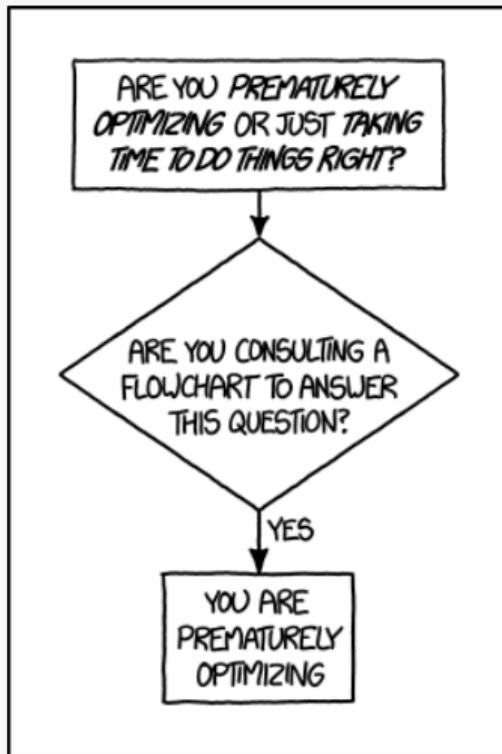
Summary

- Graph on the right compares complexity for all P223 codes.
 - Numbers in each bar indicate the number of subroutines in each category
 - Numbers at the top of each bar indicate the total number of subroutines
- Apparent that all codes have a high percentage of complex subroutines
 - Marco and MarcoAir are particularly bad



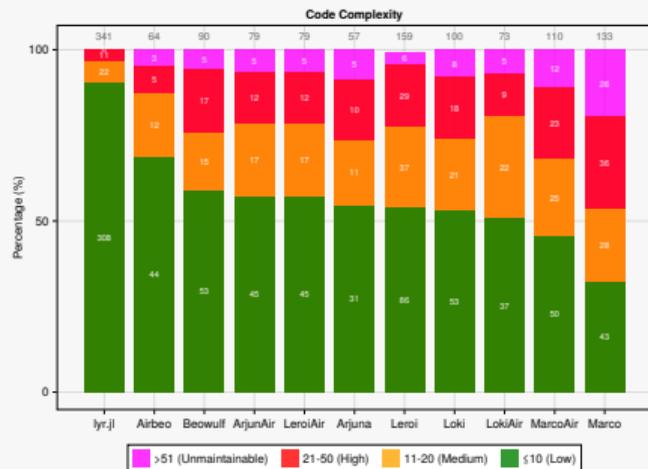
Summary

- Note has reviewed code complexity metrics
- Note documented the application of these metrics to three codes



Summary

- We can extend the comparison from three slides ago
- Immediately apparent how much simpler `lyr.jl` is
 - There is a large number of functions
 - Those functions are stored in different files
 - Most (90%) functions are low complexity
 - No unmaintainable functions
- Low complexity + simple API \implies
 - Better maintainability
 - Better basis for extension



Aegis Geophysics, 2020-02-21

References

Gao, H., H. Hijazi, J. Medeiros, J. Durães, C.-T. Lam, P. Carvalho, and H. Madeira. 2025 Complementarity in Software Code Complexity Metrics.

Halstead, M.. 1977. Elements of Software Science. Elsevier Science.

McCabe, T.. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering SE-2*, 308–320.

Raiche, A., G. Wilson, and F. Sugeng. 2007. Practical 3D inversion – The P223F Software Suite. ASEG. Perth, WA, 68–69.

Tashtoush, Y., N. Abu-El-Rub, O. Darwish, S. Al-Eidi, D. Darweesh, and O. Karajeh. 2023. A Notional Understanding of the Relationship between Code Readability and Software Complexity. *Information* 14, 81.

Wallace, D. R., A. H. Watson, and T. J. McCabe. 1996 Structured testing : A testing methodology using the cyclomatic complexity metric: Technical Report NIST SP 500-235 National Institute of Standards and Technology Gaithersburg, MD.