A brief contrast in code complexity

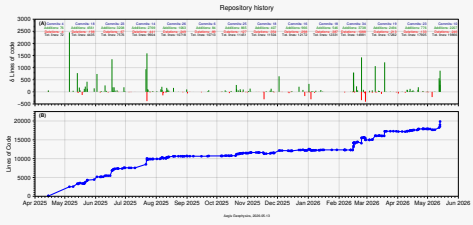
Why we need to refactor

David Annetts, Aegis Geophysics

May 13, 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

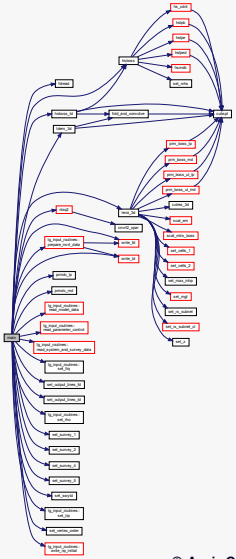


Why modernise in the first place?

- 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, managable components
  - 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` call structure. Nodes in red lead to more subroutines.



Commonality, and the lack thereof ...

- It is often the case that routines in the P223 suite with the same name are implemented differently with different functionality
- `fold_and_convolve` computes the observed response by convolving the earth response function, with the Tx waveform.
  - The AEM codes have the same complexity
  - Ground codes differ

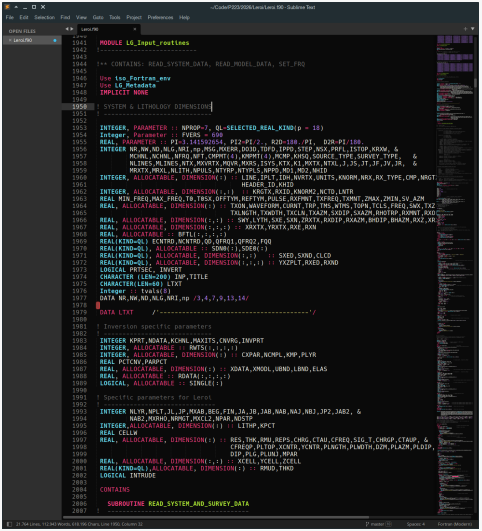
Row	function_name	complexity	Program
	String	Int64	String
1	fold_and_convolve	18	ArjunAir
2	fold_and_convolve	10	Arjuna
3	fold_and_convolve	18	AirBeo
4	fold_and_convolve	10	Beowulf
5	fold_and_convolve	18	LeroiAir
6	fold_and_convolve	14	Leroi
7	fold_and_convolve	18	MarcoAir
8	fold_and_convolve	9	Marco
9	fold_and_convolve	18	LokiAir
10	fold_and_convolve	10	Loki

The task of computing the observed response by convolving the earth response function with the transmitter waveform is common over the P223 suite.

Why can't the same code be used over the code base?

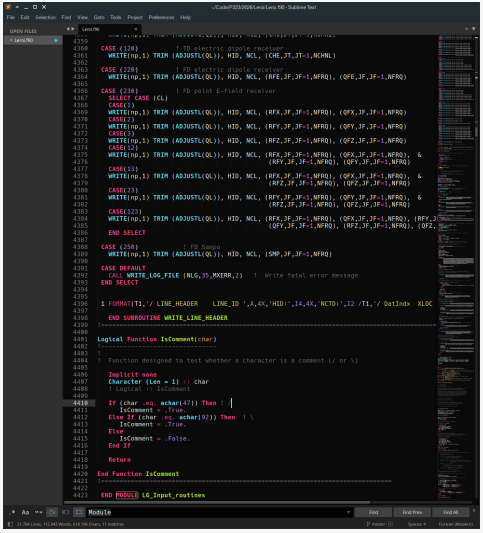
Leroi ...

- Modules
  - Note that the subroutine `Read_system_and_survey_data` has no arguments
  - Declaring all variables at the top of a module makes these variables global in any routine `use`ing that module



Leroi ...

- `lg_Input_routines` contains
  - 2482 lines
  - Six subroutines
  - One function
  - calls to 6 subroutines
- `Read_system_and_survey_data` is 827 lines of `if` blocks



We know that the P223 codes are complex.

This presentation attempts to quantify how complex they are.

Code complexity metrics

- Cognitive complexity (Gao et al., 2025)
  - A newer metric that assesses the mental effort required for a human to understand the code. It specifically penalises structures that are difficult to read, such as deep nesting, to encourage more understandable code
- Halstead complexity (Halstead, 1977)
  - 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
- Cyclomatic complexity (McCabe, 1976)
  - 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 cyclomatic complexity and note that Gao et al. (2025) recommended a holistic approach using multiple metrics

Tools

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

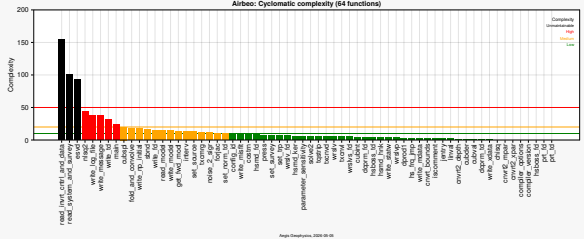
Cyclomatic complexity

- Count the branches in a routine
  - `goto, if, switch, select, ...`
- Used by Wallace et al. (1996) as a foundation for structured testing

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

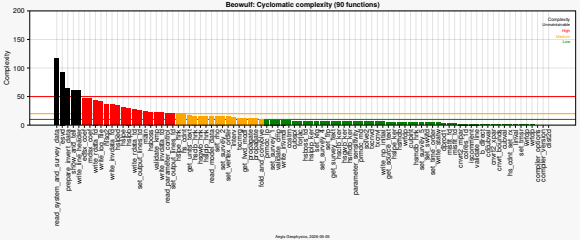
Low CC scores do not mean that code is bug-free
Low CC scores suggest that code is easier to understand and debug.
NIST recommend CC < 10; NASA recommends CC < 15
We use Wikipedia's categorisation scheme

- `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



Complexity	Total	Percentage
Simple	44	68.75
Moderate	12	18.75
High	5	7.81
Unmaintainable	3	4.69

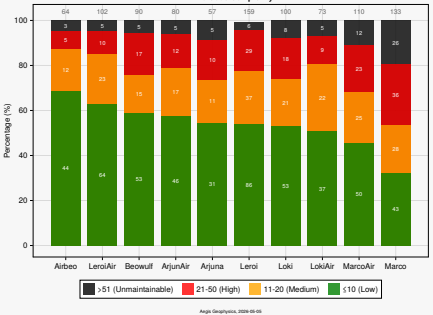
- `Beowulf` is designed to invert general EM surveys for an *n*-layered earth
  - We therefore expect an increase in complexity because of the increased number of source / receiver configurations the code has to handle
- The program contains 90 functions over 6769 lines in a single file
  - About 59% of the code is in simple functions
  - About 5.5% code in unmaintainable functions



Complexity	Total	Percentage
Simple	53	58.89
Moderate	15	16.67
High	17	18.89
Unmaintainable	5	5.56

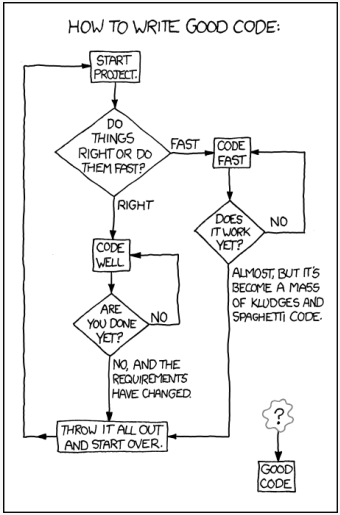
Summary

- 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



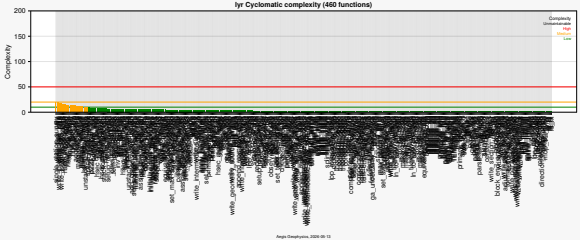
Summary

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



So what about `lyr`

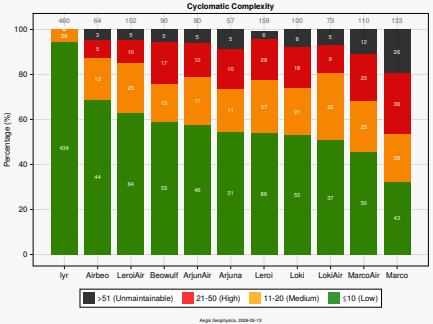
- `lyr` is designed to merge capabilities of `Airbeo` and `Beowulf`
  - Written in Julia (Bezanson et al., 2012)
  - Designed with maintainability as a prime concern
- The program contains 460 functions over 28 files with a total of 7792 lines of code
- Functional paradime encourages the use of simple functions
  - Over 90% of the code is in simple functions
  - No unmaintainable functions



Complexity	Total	Percentage
Simple	434	94.35
Moderate	26	5.65
High	0	0.00
Unmaintainable	0	0.00

Summary

- Immediately apparent how much simpler `lyr` is compared to P223 codes
  - There is a large number of functions
  - Those functions are stored in different files
  - All functions are medium or low complexity
  - No unmaintainable or high-complexity functions
- Low complexity + simple API $\Longrightarrow$
  - Easier to understand
  - Better maintainability
  - Better basis for extension



References

Bezanson, J., S. Karpinski, V. B. Shah, and A. Edelman. 2012. Julia: A Fast Dynamic Language for Technical Computing. *arXiv* 1209.5145.

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.

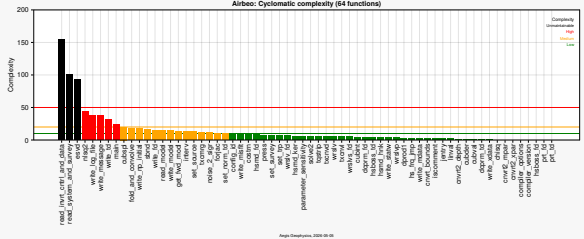
Raiche, A., G. Wilson, and F. Sugeng. 2007. Practical 3D inversion – The P223F Software Suite. ASEG. Perth, WA, 68–69.

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.

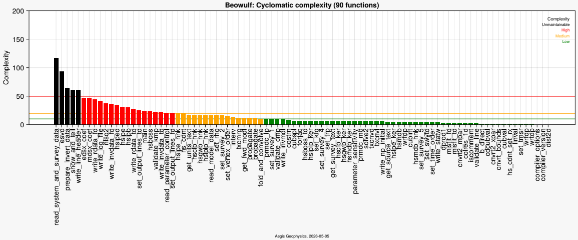
Wallace, D. R., A. H. Watson, and T. J. McCabe. 1996 Structured testing : A testing methodology using the cyclomatic complexity metric: Technical Report NIST SP 500-235 National Institute of Standards and Technology Gaithersburg, MD.

Airbeo

- `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



Complexity	Total	Percentage
Simple	44	68.75
Moderate	12	18.75
High	5	7.81
Unmaintainable	3	4.69

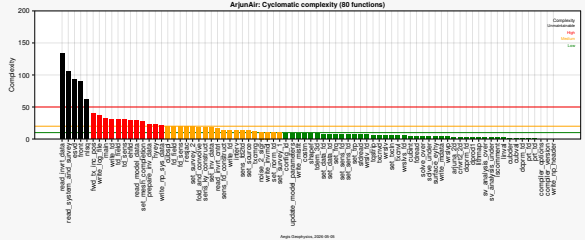


- `Beowulf` is designed to invert ground EM surveys for an *n*-layered earth

Complexity	Total	Percentage
Simple	53	58.89
Moderate	15	16.67
High	17	18.89
Unmaintainable	5	5.56

ArjunAir

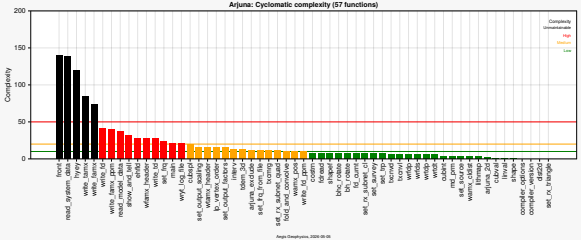
- `ArjunAir` is designed to model AEM data using a 2.5D earth modelled as 9-node isoparametric finite elements



Complexity	Total	Percentage
Simple	46	57.5
Moderate	17	12.25
High	12	15.0
Unmaintainable	5	6.25

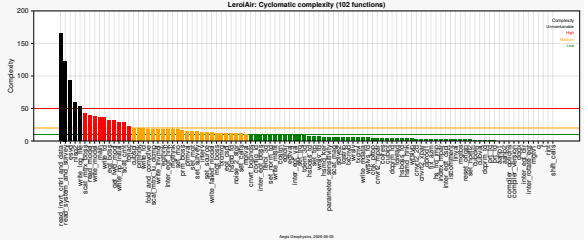
Arjuna

- `Arjuna` is designed to model data using a 2.5D earth modelled as 9-node isoparametric finite elements



Complexity	Total	Percentage
Simple	31	54.39
Moderate	11	19.30
High	10	17.34
Unmaintainable	5	8.77

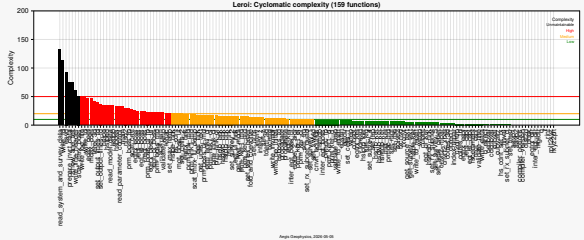
- `LeroiAir` is designed to model EM data using inductively-thin plates embedded in an *n*-layered earth



Complexity	Total	Percentage
Simple	64	62.75
Moderate	23	22.55
High	10	9.80
Unmaintainable	5	4.90

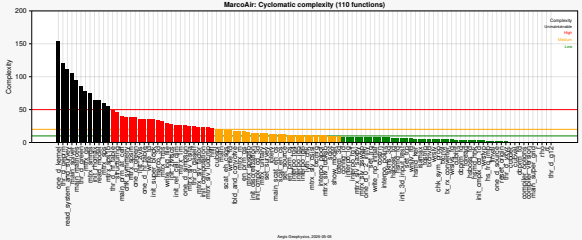
Leroi

- `Leroi` is designed to model EM data using inductively-thin plates embedded in an *n*-layered earth



Complexity	Total	Percentage
Simple	86	62.75
Moderate	37	23.27
High	29	18.24
Unmaintainable	6	3.78

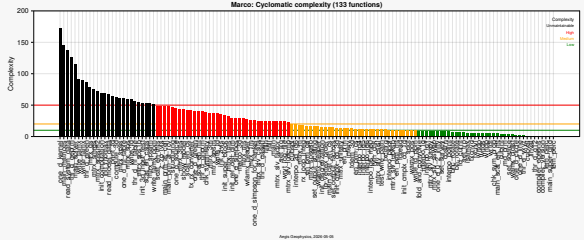
- `MarcoAir` is designed to model AEM data using rectilinear prisms embedded in an *n*-layered earth



Complexity	Total	Percentage
Simple	50	45.46
Moderate	25	22.73
High	23	20.91
Unmaintainable	12	10.91

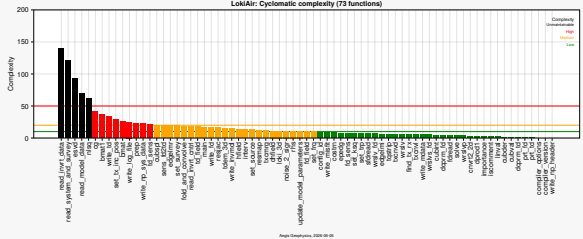
Marco

- `Marco` is designed to model EM data using rectilinear prisms embedded in an *n*-layered earth



Complexity	Total	Percentage
Simple	43	32.33
Moderate	28	21.06
High	36	27.07
Unmaintainable	26	19.55

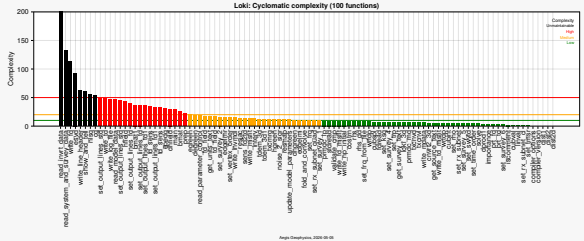
- `LokiAir` is designed to model AEM data using a 3D earth consisting of hexahedral finite elements



Complexity	Total	Percentage
Simple	37	50.69
Moderate	22	30.14
High	9	12.33
Unmaintainable	5	36.85

Loki

- `Loki` is designed to model EM data using a 3D earth consisting of hexahedral finite elements



Complexity	Total	Percentage
Simple	53	53.00
Moderate	21	21.00
High	18	18.00
Unmaintainable	8	8.00