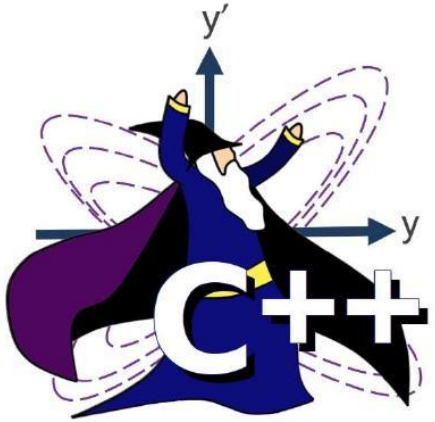
Retroactive Sustainability Improvements in the Merlin++ Particle Tracking Code

S. Rowan, S. Tygier, Y. Cai, C. Venters, R. Appleby, R. Barlow

thanks to S. Redaelli

What is Merlin++?

- Merlin++ is a multi-purpose particle accelerator and particle tracking simulation library
- Originally developed at DESY for ILC studies, circa late 90's
- Used/developed/maintained by Manchester/Huddersfield (UK) for HL-LHC / FCC collimation/loss studies since mid 2000's
- Main feature set (non-exhaustive):
  - Extensive use of Object-Orientated Design methodologies
  - Ring and beamline lattice construction (incl. MAD twiss import)
  - Single particle/bunch tracking and acceleration (optional ROOT integration)
  - Advanced collimation (conventional/HEL) and scattering (pomeron)
  - Wakefield simulations
  - and many more…



What is Software Sustainability?

Example of bad practice (…and why Merlin++ is a good case study!)

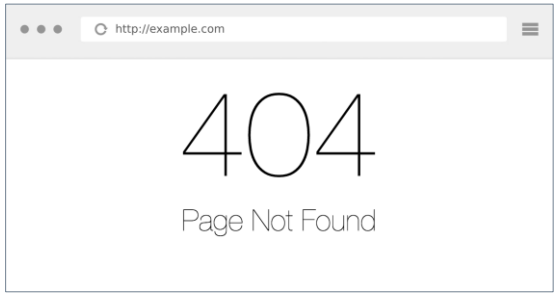
Start working on software

Look for documentation

find none...

Look through old literature

find reference to website!!



Wayback to the rescue!

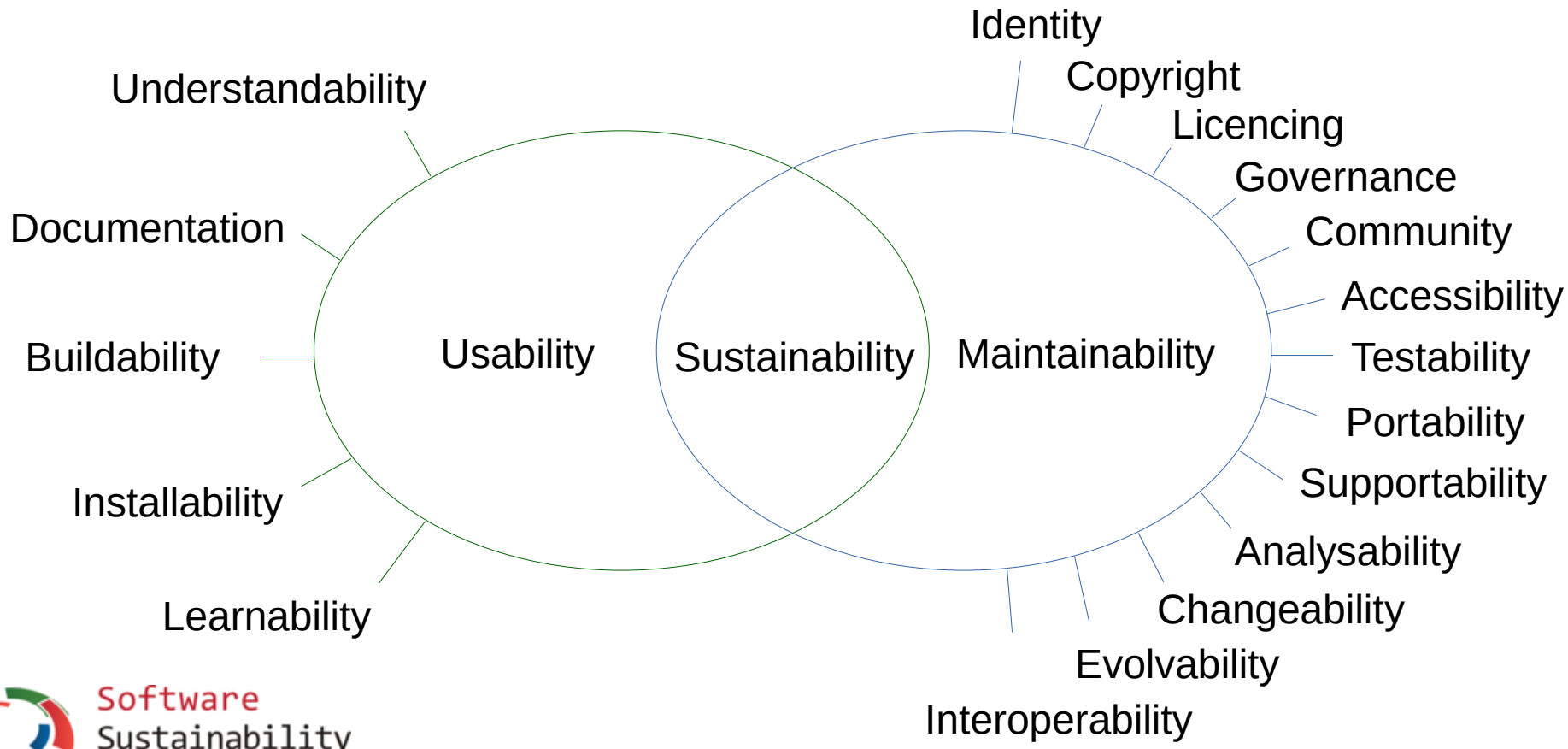
INTERNET ARCHIVE
WayBackMachine

Genuine quote!

“Unfortunately, what is sadly lacking is any form of general documentation for the library (e.g. a user’s guide.)”

What is Software Sustainability?

A more formal definition (though not the only one!)



Software Sustainability Institute

SSI Sustainability Evaluation

- The UK Software Sustainability Institute (link) identifies two forms of sustainability assessment:
  - **Tutorial-based Assessment** → Focus on user/developer experience
    - Qualitative
  - **Criteria-based Assessment** → Focus on meeting specific criterion
    - Quantitative

- Decision to focus on criteria for improvements
- Note that the SSI assessments focuses on a software package's surrounding infrastructure rather than code base itself → a good place to start nonetheless

SSI Sustainability Evaluation

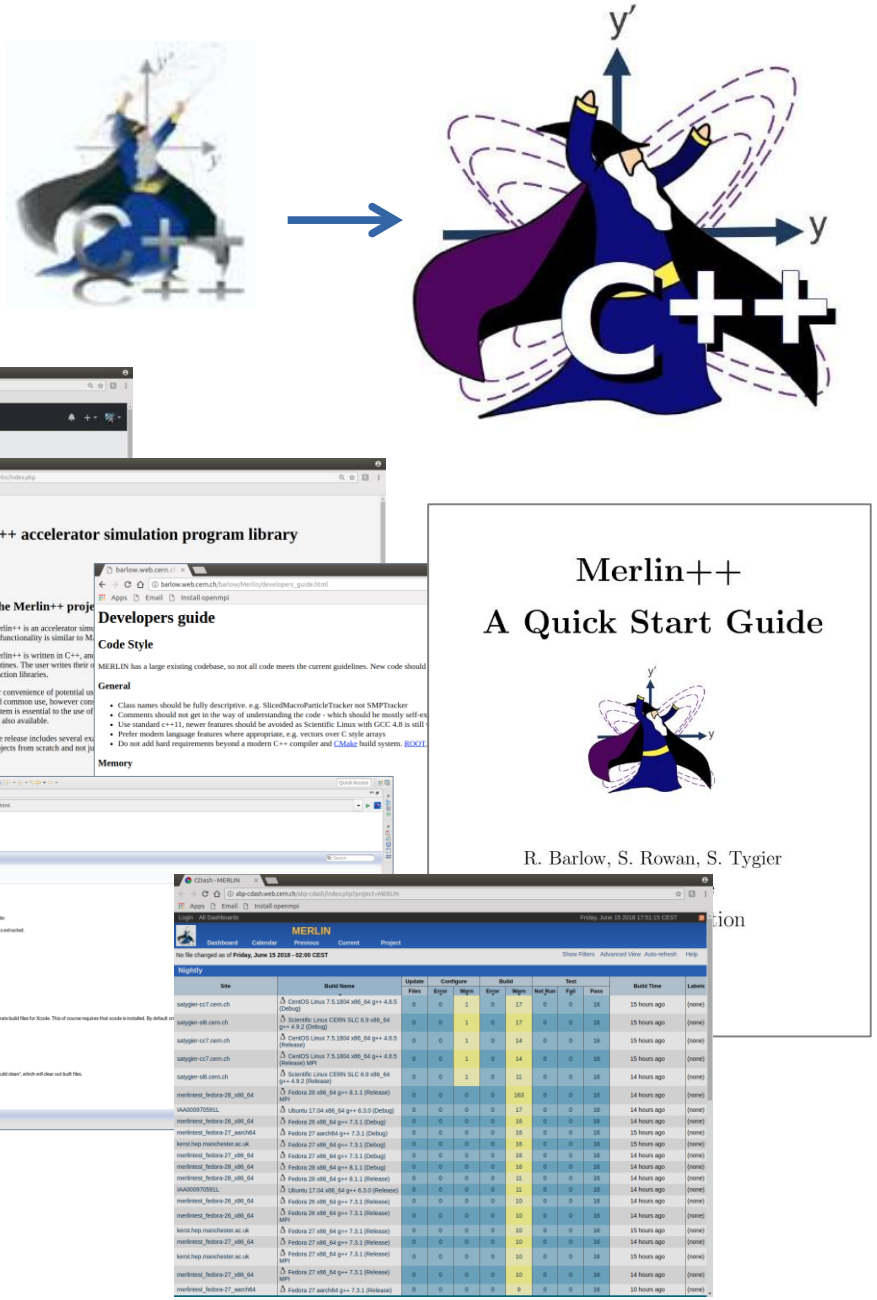
- **Criteria-based Assessment**
  - Originally generally poor / unsatisfactory
  - A *lot* of work required!
  - Overall, significant improvements throughout!

Evaluation Key:
0-25% → Poor
25-50% → Unsatisfactory
50-75% → Satisfactory
75-100% → Excellent

Sustainability Metric	Met Criterion	Original Evaluation	Met Criterion	New Evaluation
Understandability	3/7	Unsatisfactory	6/7	Excellent
Documentation	3/19	Poor	14/19	Satisfactory
Buildability	3/9	Unsatisfactory	8/9	Excellent
Installability	7/14	Unsatisfactory	9/14	Satisfactory
Learnability	0/5	Poor	3/5	Satisfactory
Identity	3/7	Unsatisfactory	5/7	Satisfactory
Copyright	1/5	Poor	5/5	Excellent
Licencing	3/4	Satisfactory	4/4	Excellent
Governance	1/2	Unsatisfactory	2/2	Excellent
Community	1/11	Poor	6/11	Satisfactory
Accessibility	6/11	Satisfactory	8/11	Satisfactory
Testability	1/17	Poor	11/17	Satisfactory
Portability	10/16	Satisfactory	10/16	Satisfactory
Supportability	4/19	Poor	10/19	Satisfactory
Analysability	6/16	Unsatisfactory	13/16	Excellent
Changeability	3/10	Unsatisfactory	9/10	Excellent
Evolveability	0/3	Poor	2/3	Satisfactory
Inoperability	2/3	Satisfactory	3/3	Excellent

Addressing Criteria

- **Usability**
  - → New Website (soon™)
  - → Clean/Public Github repository (link)
  - → Build/Install/IDE use guide
  - → Quick Start Guide/Examples/Tutorials
  - → Full User Guide being drafted
- **Maintainability**
  - → MERLIN → Merlin++
  - → API/Class library documentation (doxygen)
  - → Practical test suite/Nightly builds (cdash)
  - → Copyright standardization to GPL2+
  - → Standardized code style/formatting (Uncrustify)
  - → Developer/Coding style guide/pre-commit hooks



Code Base Analysis

Code Quality Analysis

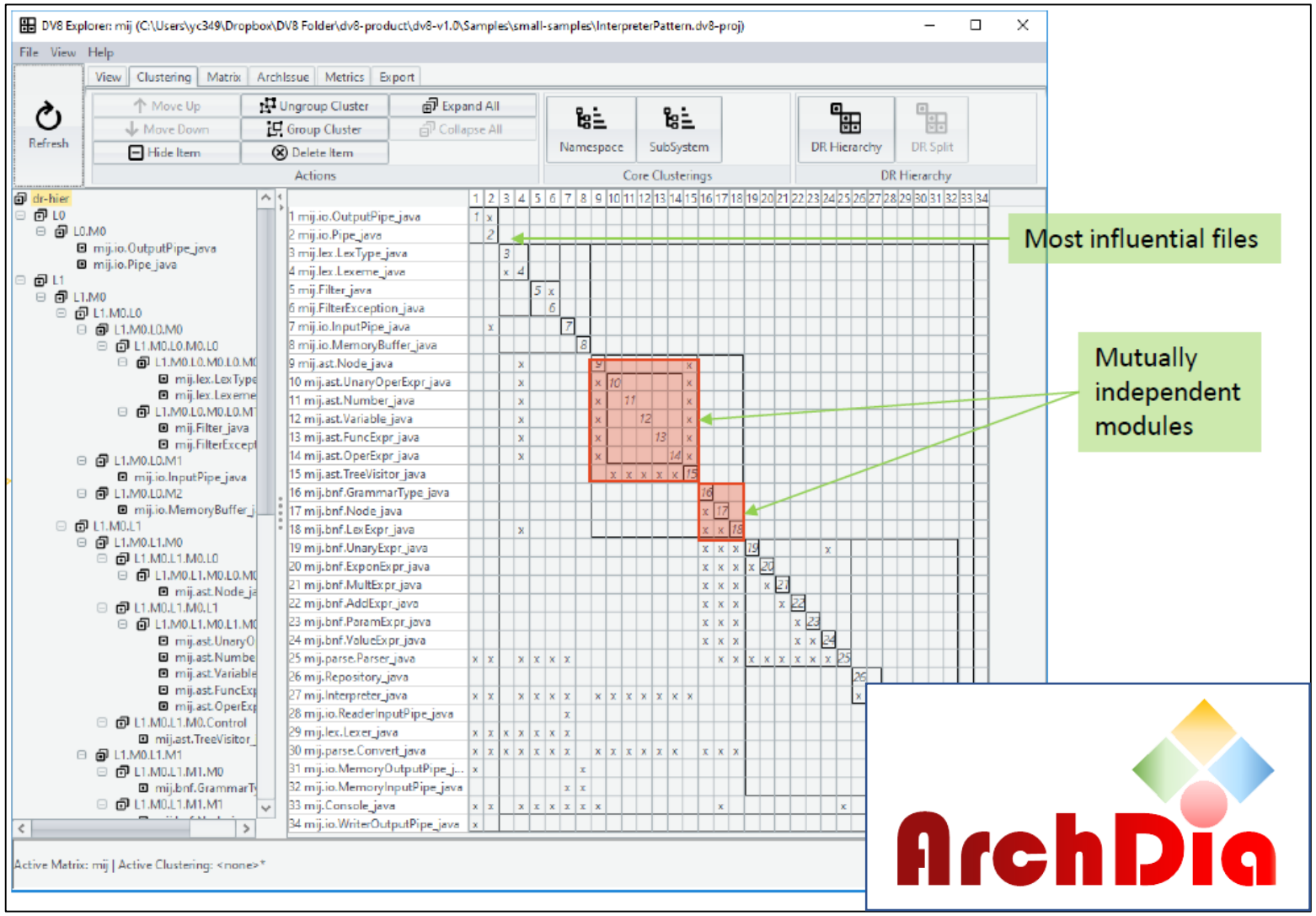
- To identify technical debt arisen in terms of code quality we used the following
  - Static Analysis: Eclipse CDT plugin ‘Metriculator’ → Complexity/LSLOC/Efferent Coupling etc
  - Dynamic Analysis: Valgrind, Intel’s VTune Amplifier → memory leaks/cache misses etc

Architecture/Structural Analysis

- In collaboration with Drexel University to analyze Merlin++ using their **ArchDia DV8** tool suite (link). Constructs **design structure matrices** (DSMs) to quantify architectural debt:
  - *Decoupling Level*
  - *Propagation Cost*
  - *Package Cycling*
  - *Unhealthy Inheritances*
  - *and many more…*

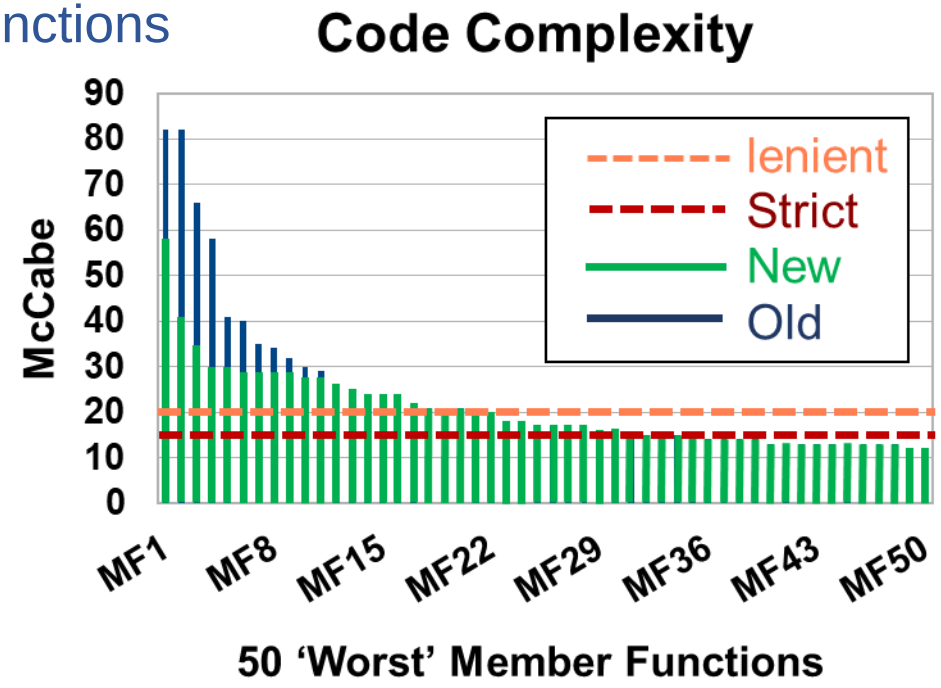
Code Base Analysis

- A little more on DV8…
- The ‘Design Rule Hierarchy’
- Robust clustering algorithms identify most influential files (files in low layer groups depend on those higher up)
- Files within layer groups are ordered into mutually independent modules
- Process allows one to identify dependency hotspots and coupling issues



Code Quality Analysis

- Code quality analyses identified ~30 ‘severe’ classes/member functions which require reworking, i.e. exceeded lenient limits on at least 1 metric
  - For perspective, Merlin++ contains 5332 member functions
- Investigation revealed common **code smells**
  - Functions trying to do too much ‘God Objects’
  - C/MATLAB-style code/High Complexity
    - Mass use of for loops/if/switch statements etc by less-skilled, more recent user-developers
    - Focus on functionality and simulation results rather than code design
  - *Solution: OOD/Design Patterns/S.O.L.I.D*

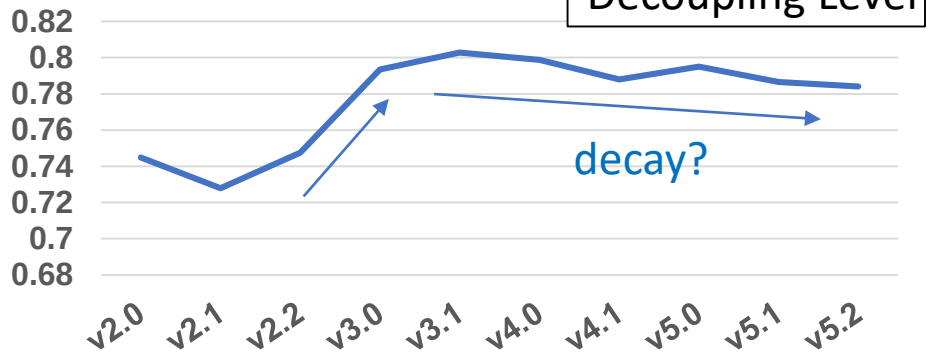


- Subsequent dynamic analysis showed that addressing complexity issues inherently improved performance 2-3% by means cache miss reduction

Architecture/Structural Analysis

- DV8 architecture analysis revealed Merlin++ to be somewhat structurally sound (great news?)
  - Relatively high (~0.78) decoupling level
  - Relatively low (~0.11) propagation cost
- Nonetheless, a number of architectural flaws were identified – likely responsible for propagated technical debt
  - Package Cycling (12 violations)
  - Unhealthy Inheritance (18 violations)
- We found that the clustered DSM analyses not only identified flaws, but provided clear solution pathways

Decoupling Level



Unhealthy Inheritance

		1	2	3	4	5
/PSmoments.h	1	*1*				
/SliceMacroParticle.h	2	T,P,U	*2*	C		
/SliceMacroParticle.cpp	3	T,C	I,I,U	*3*		
/SMPBunch.h	4	T,U	I,T,U		*4*	
/SMPBunch.cpp	5	C,T	T,C	C	M,T,I,C,I,U,R	*5*

Violates Liskov Substitution Principle

Package Cycling

		1	2	3	4
/Aperture.h	1	*1*			
/Aperture.cpp	2	T,I,O,I,C,U	*2*	I	
/CollimatorAperture.h	3	I,P	O	*3*	O
/CollimatorAperture.cpp	4	R,C,U	O,C	S,I,R,I,U	*4*

Summary and Conclusions

- Merlin++ particle tracking software was noted to have significant user-developer issues
- UK Software Sustainability Institute criteria-based assessment was used to identify and implement Merlin++ accessibility, usability and maintainability improvements
- Code quality analyses were carried out using Metriculator/Valgrind/Vtune Amplifier
- Structural dependency/architectural analysis was carried out using ArchDia DV8
- The Merlin++ developers found their approach to improving sustainability to be very effective
- Cyclomatic Complexity and Structural Dependency (DSM) analyses were found to be particularly useful

Thanks for listening!