

CAS 741 (Development of Scientific Computing Software)

Winter 2024

Assurance Cases

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 19, 2024



Assurance Cases

- Administrative details
- Final documentation
- Make
- Coding standards
- Coding advice
- Connecting code to MG and MIS
- License and copyright
- README file
- Other files in capTemplate
- Assurance cases

Administrative Details

- Draft participation grade
- When developing your code, remember that your goal is for someone else to be able to compile and run it
- Upcoming classes
 - ▶ L19 — Assurance Cases
 - ▶ L20 — Artifact Generation
 - ▶ L21a–24 — Implementation/Testing Presentations
 - ▶ L21b — A Holistic Approach (not this year)
- No requirement to provide feedback to colleagues on final documentation

Administrative Details: Report Deadlines

Final Documentation Week 13 Apr 12

- The written deliverables will be graded based on the repo contents as of 11:59 pm of the due date
- If you need an extension for a **written** doc, please ask
- When ready, assign issues to your primary and secondary reviewers
- GitHub issues due two days after assignment deadlines
- From Drasil Code onward, Drasil projects no longer need to maintain traditional SRS

Administrative Details: Presentations

Unit VnV/Implement Week 12 Week of Apr 3

- Specific schedule depends on final class registration
- Informal presentations with the goal of improving everyone's written deliverables
- Domain experts and secondary reviewers (and others) will ask questions

Presentation Schedule

Presentation Sched Cont'd

- Implementation Present (15 min each)
 - ▶ **Mar 26: Reyhaneh, Waqar, Al, Tanya, Atiyeh**
 - ▶ **Apr 2: Nada, Phil, Xinyu, Fasil, Yi-Leng**
 - ▶ Apr 5: Gaofeng, Morteza, Valerie, Hunter, Ali
 - ▶ Apr 9: Cynthia, Adrian, Yiding, Kim Ying

Presentation Schedule

- 3 presentations each
 - ▶ SRS everyone
 - ▶ VnV and POC subset of class
 - ▶ Design subset of class
 - ▶ Implementation everyone
- If you will miss a presentation, please trade with someone
- Implementation presentation could be used to run a code review, or code walkthrough

Final Presentations

- Based on your implementation and testing
- You decide what to focus on
- If in doubt, ask
- Options
 - ▶ Demonstration
 - ▶ Summarize testing results
 - ▶ Usability test with the class (show of hands for data)
 - ▶ Code walkthrough
 - ▶ Technology overview
 - ▶ CI/CD
 - ▶ Valgrind
 - ▶ Doxygen
 - ▶ etc.

Final Documentation

- Looking for
 - ▶ Revision of documentation
 - ▶ Consistency between documents
 - ▶ Traceability between documents - should be able to pick a requirement and trace it all the way to testing
 - ▶ Effort made to address issues and comments
 - ▶ Appropriate challenge level
- Make it easy to see changes from Rev 0
 - ▶ Reflection document
 - ▶ Closed issues in the issue tracker
 - ▶ Specific explanation in Revision History
 - ▶ Comments in tex file

Final Documentation

- Faking a rational design process
- Problem Statement revised and improved
- Requirements Document revised and improved
- Design Documents revised and improved
- VnV Plan revised and improved (complete unit testing sections)
 - ▶ Summarize unit testing philosophy
 - ▶ Point to unit testing code
- VnV Report
- Source Code
- Drasil projects no longer need to maintain the traditionally generated SRS
- Reflection Document

Final Doc: Reflection

- Reflection document updated in [capTemplate repo](#)
- Summarize changes in response to feedback from reviewers, instructor, supervisor, users
- Include hyperlinks to relevant closed issues
- Explain how you arrived at your final design and implementation
- Reflect on and justify your design decisions
- Ignore questions related to:
 - ▶ Hazard analysis
 - ▶ Economic considerations
 - ▶ Feedback on capstone
- How did your project management compare to your dev plan
 - ▶ What went well? (continue)
 - ▶ What went wrong? (stop)
 - ▶ What would you do differently next time? (start)

Final Project Quality

- Installability - instructions given, makefiles etc to support, means to validate the installation, required libraries are explicitly identified
- Learnability - instructions to get someone started using the software
- Robustness - can the software handle garbage inputs reasonably
- Performance - measured if appropriate
- Usability - measured if appropriate

Installability and Learnability

- You can test this
- Ask a colleague to install your software
- Run it on a virtual machine, like [VirtualBox](#)
- Use a “light weight” VM like docker
- Include installation instructions (INSTALL.txt)
- Include instructions so that someone else can run your tests cases
- Part of the evaluation of each project is to run it

Consider Make for Installability, running test cases

- Tutorial on Make, with links
- Example Makefile for GlassBR

Unit VnV Plan

- Complete VnV Plan
- Scope - what modules will be verified
- Your approach for automated testing (if not already covered)
- Tools for code coverage metrics (if not already covered)
- Non-testing based verification (if planned)
- Unit test cases for each module - from black box and white box (can point to code)
- Performance tests for individual modules (if appropriate)
- Evidence that all modules are considered

Final Documentation: VnV Report

- Completing what you proposed in your test plan
- You do not need to repeat material from your test plan - the emphasis is not on the rationale for test case selection, but on the results.
- If your test plan does not match what you are now testing, edit your test plan to “fake” a rational design process.
- If your test report is not complete, because there is not time for all of the tests, explain this in your report

VnV Report Continued

- Point to specific test cases in test plan
- Summarize your test results
 - ▶ Test case name
 - ▶ Initial state
 - ▶ Input
 - ▶ Expected results
 - ▶ Whether actual output matched expected
- Summarize and explain usability tests - quantify the results
- Performance tests - quantify the results
- Stress tests
- Robustness tests
- After quantification of nonfunctional tests, explain significance of results

VnV Report Continued

- In cases where there are many similar tests
 - ▶ Summarize the results
 - ▶ If the expected result is obvious, you might not need to state it
 - ▶ Give an example test case, and explain how similar tests were constructed
 - ▶ If the tests were random, describe how they were selected, and how many, but not all of the details
 - ▶ Use graphs and tables
 - ▶ You need enough information that
 - ▶ Someone could reproduce your tests
 - ▶ Your test results are convincing
 - ▶ Evidence that you have used testing to improve the quality of your project

VnV Report Continued

- Summarize changes made in response to test results
- Explain your automated testing set-up (if require more detail than from the test plan)
- Provide traceability to requirements (if not in test plan)
- Provide traceability to modules (if not in test plan)
- Make sure you show test results for “bad/abnormal” input

Sample VnV Report Documents

- Screenholders
- 2D Physics Based Game (Uses doxygen)
- Capstone Sample reports
- Solar Water Heating System
- Follow given template
- Examples are not perfect
- Examples are intended to give you ideas, not to be strictly followed
- You can modify/extend the test report template as appropriate

Final Documentation: Source Code

- Source code in src folder
- Comments on “what” not “how”
- Identifiers that are consistent, distinctive, and meaningful
- Avoidance of hard-coded constants (other than maybe 0 and 1)
- Appropriate modularization
 - ▶ Follow module guide
 - ▶ Show traceability between MG modules and code files
- Consistent indentation
- Explicit identification of coding standards (see next slide)
- Parameters are in the same order for all functions
- Descriptive names of source code files
- Show mapping between MIS symbols and code symbols

Coding Style

- Having a coding standard is more important than which standard you use
- Examples
 - ▶ Google guides
 - ▶ Python
 - ▶ C++
 - ▶ Java
 - ▶ NASA C Style Guide
- Important to be consistent

Doxygen

- A tool that generates documentation (say in html or tex) from the code
- Comments with special syntax are used in source files to mark information for Doxygen to use
- [Tutorial on Doxygen](#)
- There are alternative to doxygen (pydoc, javadoc, sphinx, etc.)

No License?

- Can others use your work if you do not include a license?
- [See this link for the answer](#)

Copyright

- Your work is automatically afforded protection by copyright law
 - ▶ You cannot infringe on someone else's copyright
 - ▶ Must be some creativity
- Additional protection through registration with the copyright office
- Copyright does not apply to the idea, but the expression of the idea
- Trademarks and patents cover concepts and ideas
- In work for hire, copyright belongs to employer
- You can assign your copyright to someone else or a corporation

Rights

- Owner has full and exclusive rights to control who may copy or create a derivative work
- Right to sue for copyright infringement

Licensing

- Permission to others to reproduce or distribute a work
- Licenses are distinguished by the restrictions (conditions)

Proprietary License

- Copyright holder retains all rights
- Cannot copy
- Cannot use
- Cannot modify

GNU General Public License (GPL)

- Can copy the software
- Can distribute the software
- Can charge a fee to distribute the software (which will still include the license information)
- Can make modifications
- Condition – all modifications/uses are also under GPL, source code must be available
- Lesser GPL allows to link to libraries, without automatically falling under GPL conditions

BSD and MIT

- Removes “virus” from GPL
- Can copy, distribute, charge a fee, make modifications
- Under the condition that you keep the license intact, credit the author
- Not required to disclose source
- Use at your own risk (cannot sue)

Public Domain

- Do what you want with the code
- No conditions

Copyright and License Related Links

- [Developer's guide to copyright law](#)
- [Summary of licenses](#)
- [Main types of licenses](#)
- [Choose a license](#)
- [Another summary](#)
- [Plain English summaries](#)

Other Potential Files in Your Project

- README
- Contributing guidelines
- Citation
- Changelog
- Install/Uninstall
- Dependency list
- Authors
- Code of conduct
- Acknowledgements
- Style guide
- Release information
- Product roadmap
- Getting started guide, user manual, tutorials
- FAQ

Code of Conduct

- Open source projects with a large diverse base of developers frequently create a code of conduct
- Open Code of Conduct
- Diversity statement for Python
- Geek Feminism

README files

- Make sure the README file on your landing page is up to date
- Categories and Contents of README files
- A README file is useful in any folder
- Give the reader information on the contents of the folder

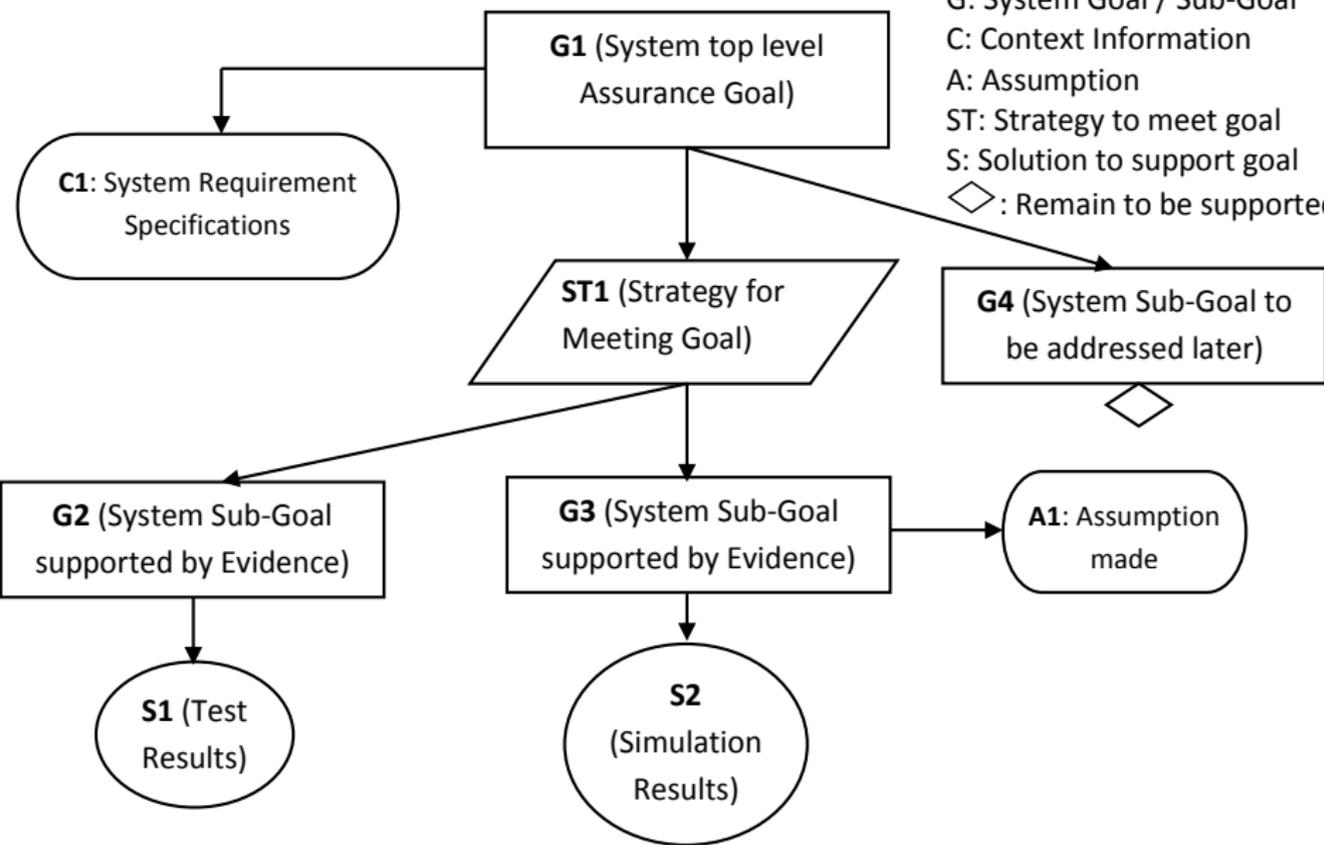
Assurance Cases in Scientific Computing [2, 1]

- Assurance cases
 - ▶ Organized and explicit argument for correctness
 - ▶ Successfully used for safety critical systems
- Advantages for SC
 - ▶ Engaging domain experts
 - ▶ Producing necessary and relevant documentation
 - ▶ Evidence that can be verified/replicated by a third party
- Example of 3dfim+
 - ▶ No errors found
 - ▶ However
 - ▶ Documentation ambiguities
 - ▶ No warning about parametric statistical model

Assurance Cases in SC Motivation

- Do we put too much trust in the quality of SCS?
- Are enough checks and balances in place, especially for safety related software?
- Problems with imposing external requirements for certification
 - ▶ External body does not have expertise
 - ▶ SCS developers dislike documentation
- Solution – Assurance Cases by experts
 - ▶ Experts engaged
 - ▶ Relevant documentation
- Current techniques of development and testing still used, but arguments will no longer be ad hoc and incompletely documented

G: System Goal / Sub-Goal
C: Context Information
A: Assumption
ST: Strategy to meet goal
S: Solution to support goal
◇: Remain to be supported



[A] AFNI: tmp/LRtap/mdef3d_01+ori

[order: RAI=DICOM]
 x = -39.500 mm [R]
 y = 31.500 mm [P]
 z = 45.500 mm [S]

Xhairs Multi X+

Color

Gap Wrap

Index

Axial

Sagittal

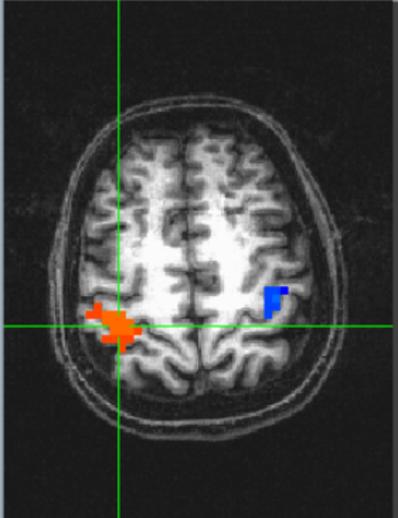
Coronal

Original View
 AC-PC Aligned
 Talairach View

See Markers

See OverLay

[A] AFNI: tmp/LRtap/m



Colr

c

b

r

g

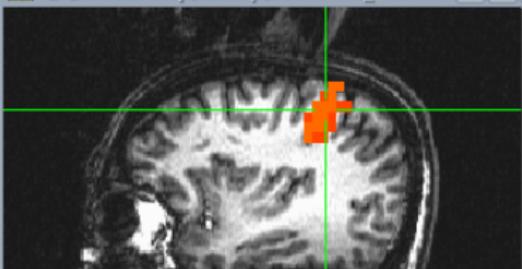
i

9

Z

141

[A] AFNI: tmp/LRtap/mdef3d_01+ori



Colr

c

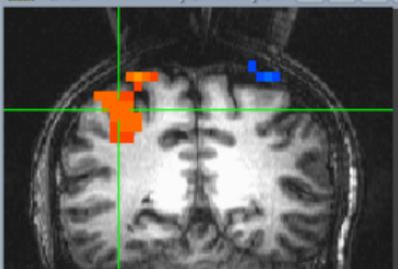
b

r

g

i

[A] AFNI: tmp/LRtap/m



Colr

c

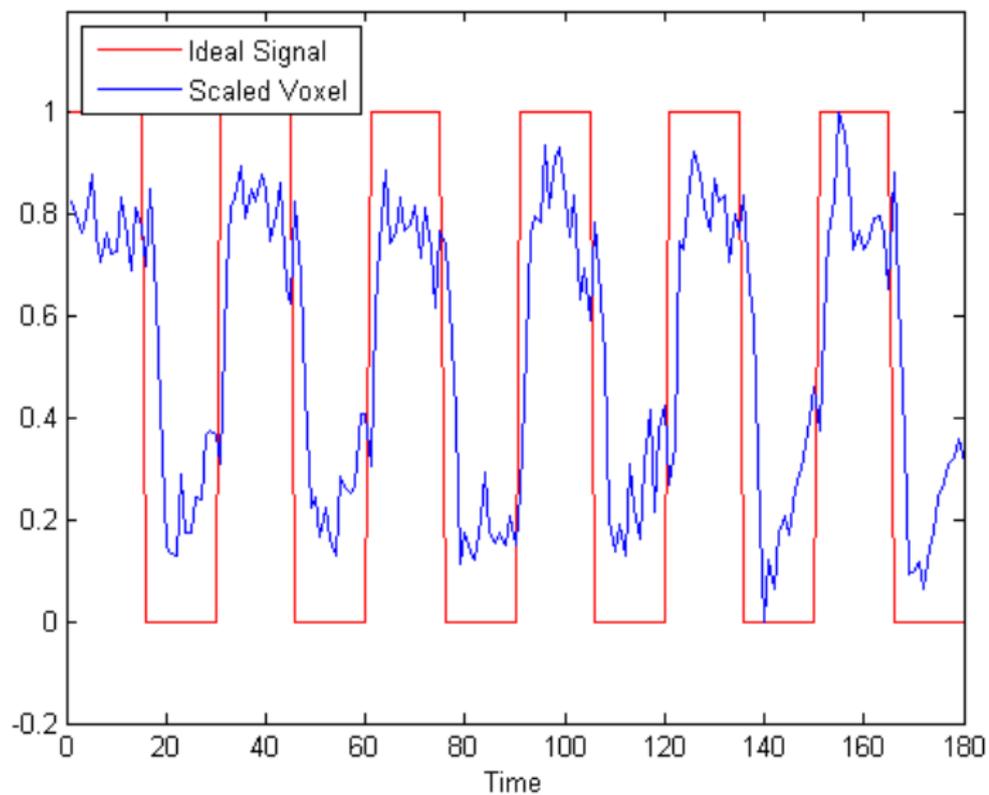
b

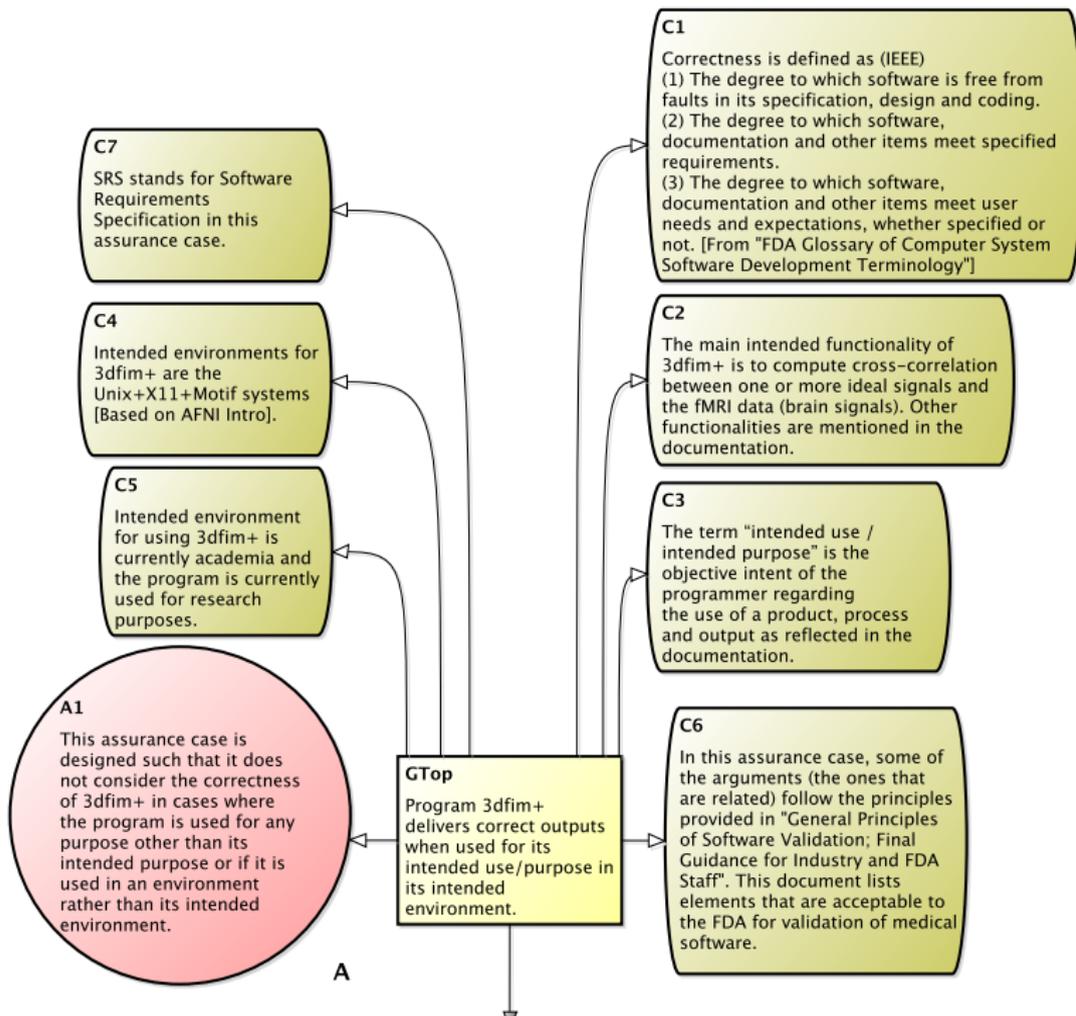
r

g

i

Scaled Voxel (23,27,22) and Ideal Signal over time





G_{Top}

Program 3dfim+ delivers correct outputs when used for its intended use/purpose in its intended environment.

S_{Top}

G can be decomposed into:

GR. 3dfim+ requirements are documented and documentation of the requirements is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.
GD. The design of 3dfim+ complies with its requirements and it is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.
GI. The implementation of 3dfim+ complies with its requirements and it is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.
GA. Inputs to 3dfim+ satisfy the defined operational assumptions.

Reasoning Proof:

Premise: GR, GD, GI and GA are true.

Conclusion: G_{Top} is valid.

J_{Top}

The major software development lifecycle steps are: Requirements, Design and Implementation with appropriate V&V activities. V&V activities will be reflected in claims regarding validation of requirements, and verification of design and implementation. If requirements are appropriate, and design and implementation are appropriate and they comply with the requirements, then 3dfim+ will have been shown to deliver correct outputs. Moreover, as meeting the input assumptions is of great importance, it is considered as a separate goal; however, the correctness, completeness and consistency of the assumptions have been shown in the GR as a part of the requirements correctness, completeness and consistency.

GR

3dfim+ requirements are documented and documentation of the requirements is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.

GD

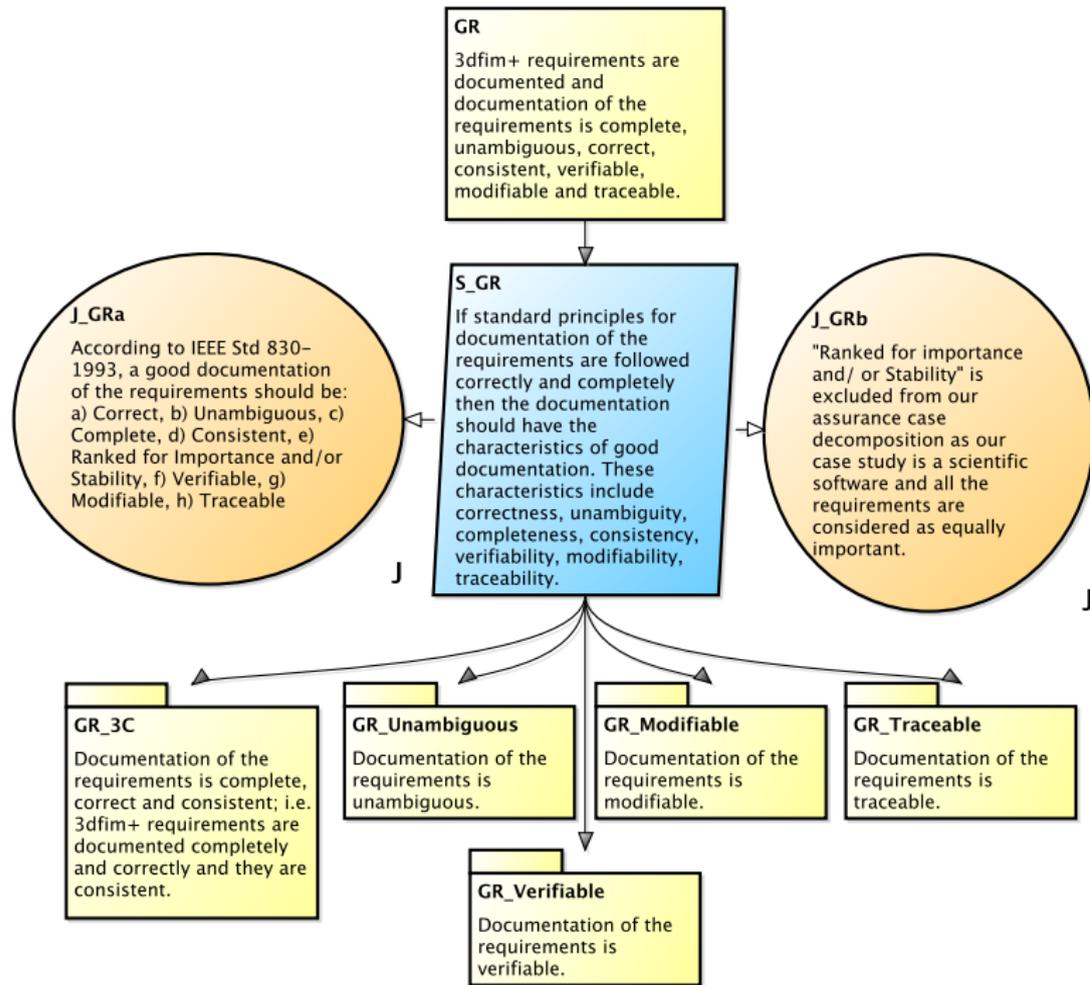
The design of 3dfim+ complies with its requirements and it is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.

GI

The implementation of 3dfim+ complies with its requirements and it is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.

GA

Inputs to 3dfim+ satisfy the defined operational assumptions.



C_ModifiableA

According to IEEE Std 830-1993, a documentation of the requirements is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires a requirement documentation to

- a) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing.
- b) Not be redundant; the same requirement should not appear in more than one place in the documentation.
- c) Express each requirement separately, rather than intermixed with other requirements.

GR_Modifiable

Documentation of the requirements is modifiable.

Modifiable.1

The SRS has a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing.

Modifiable.2

There is no duplication between the requirements.

Modifiable.3

Each requirement is expressed separately, rather than intermixed with other requirements.

S_Modifiable.1

If a standard / correct well-structured template has been followed by a competent team, then the documentation is structured and presented correctly.

S_Modifiable.2

there is no specified approach or tool for checking duplication in a document, hence a review must be done manually by the experts/ developers.

S_Modifiable.3

there is no specified approach or tool for checking this matter, hence a review must be done manually by the experts/ developers.

Modifiable.1.1

A standard / correct well-structured template has been followed.

Modifiable.1.2

The template has been followed by a competent team.

Modifiable.1.3

The documentation has been reviewed by the domain experts to make sure the template has been followed correctly.

Modifiable.2.1

The documentation has been reviewed by domain expert to make sure there is no duplication between the requirements.

Modifiable.3.1

The documentation has been reviewed by domain expert to make sure each requirement is atomic.

C_ModifiableB

List of the team members.

E_Modifiable.1

The standard template.

E_Modifiable.2

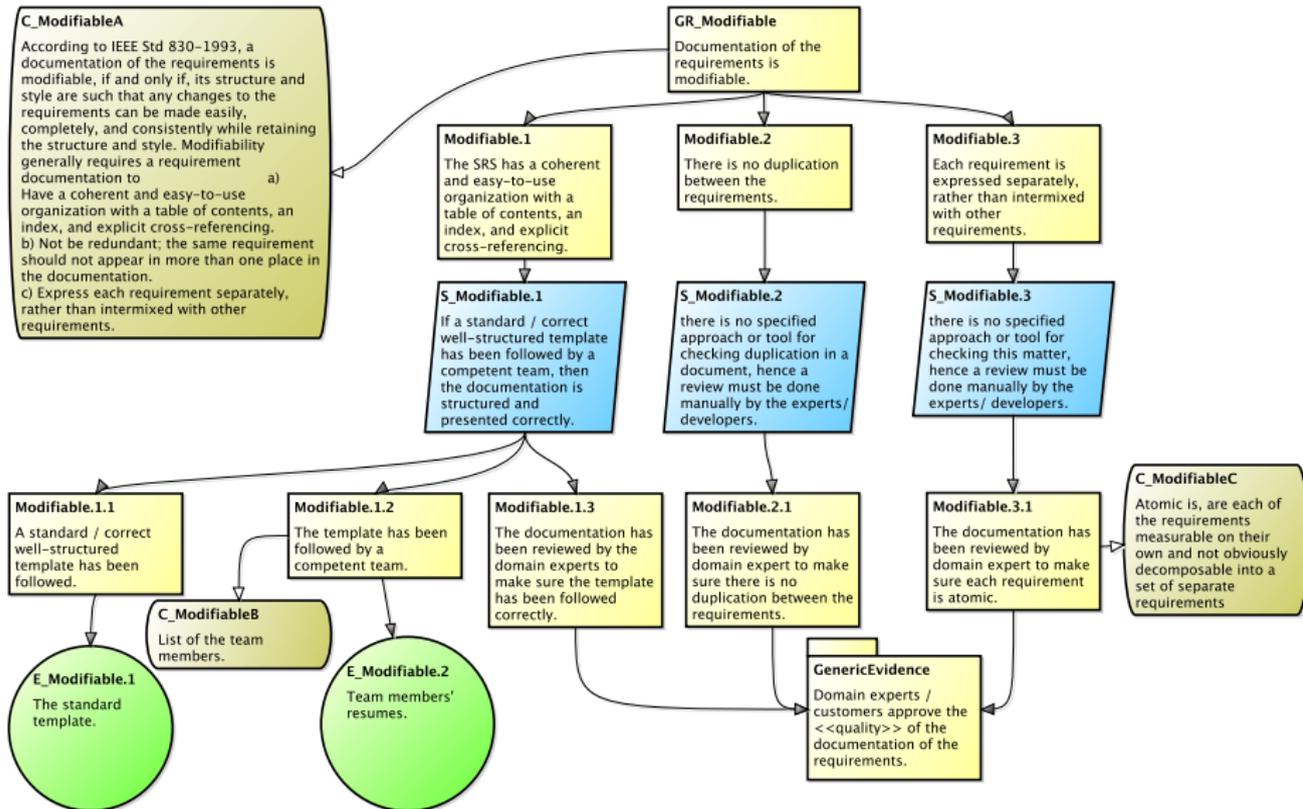
Team members' resumes.

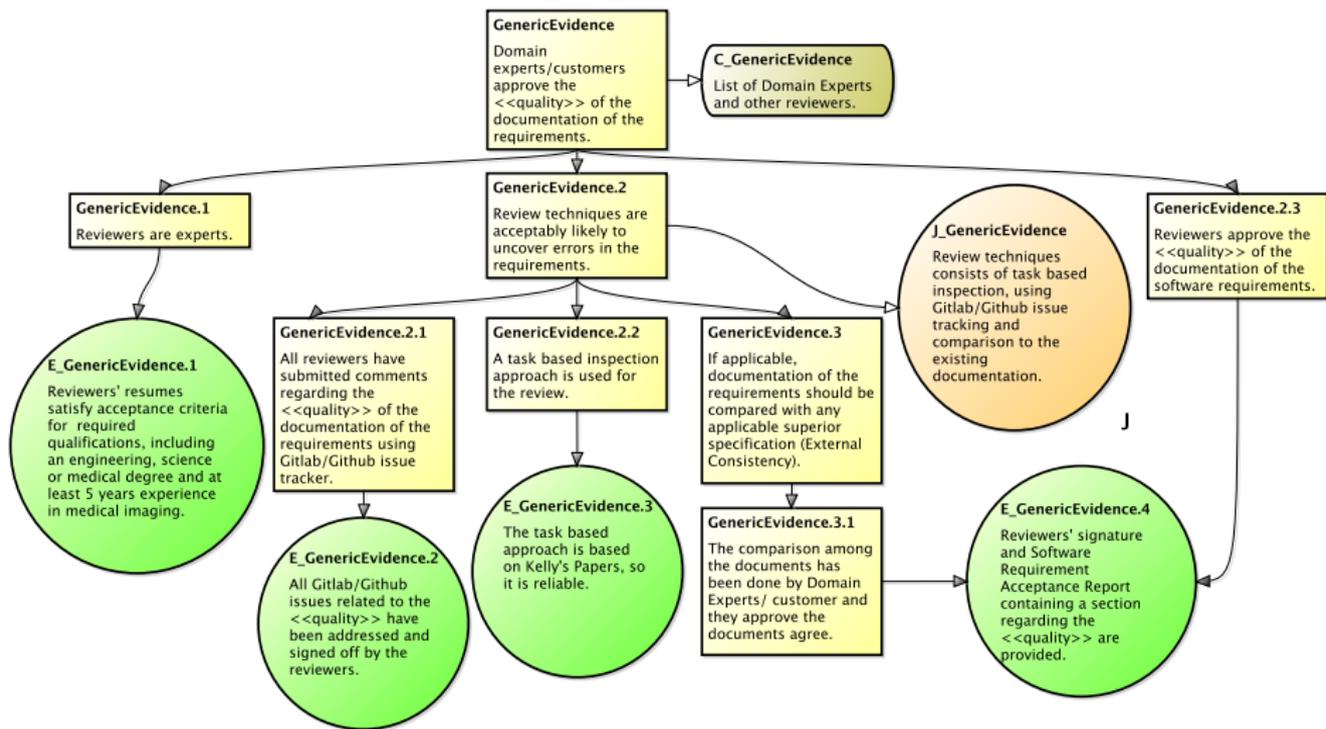
GenericEvidence

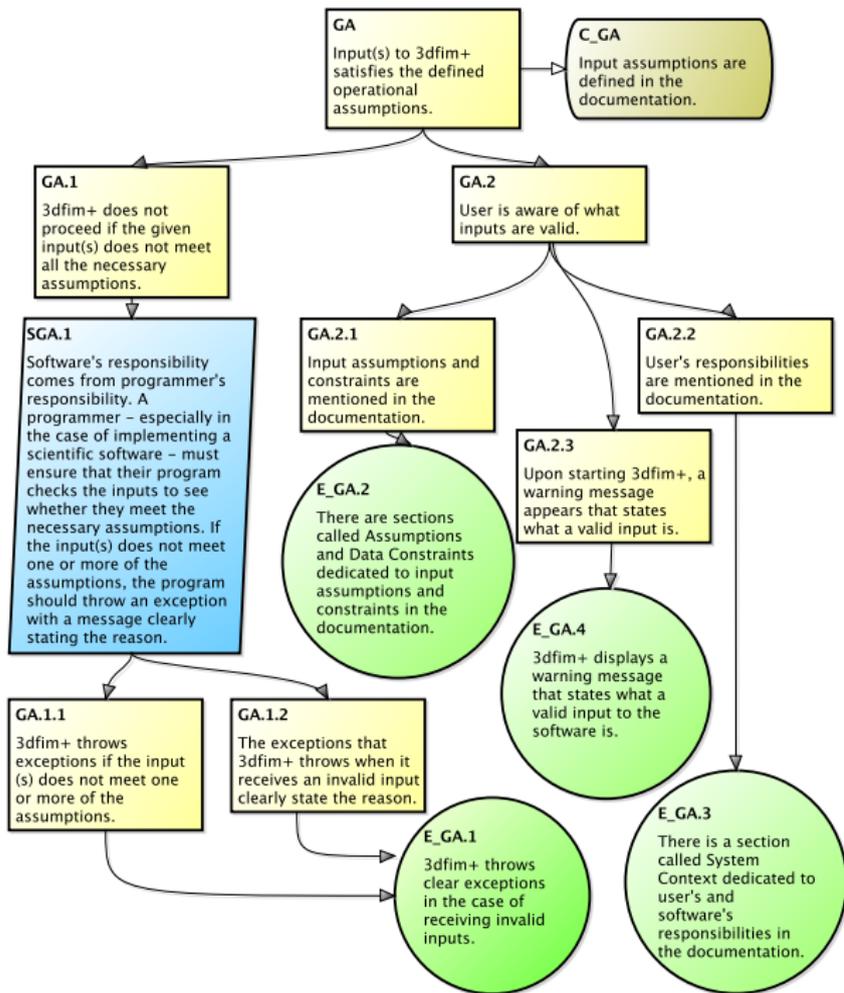
Domain experts / customers approve the <<quality>> of the documentation of the requirements.

C_ModifiableC

Atomic is, are each of the requirements measurable on their own and not obviously decomposable into a set of separate requirements







Proposed Changes to 3dfim+

- No mistakes found in calculations
- Goal of original software was not certification
- Problems found
 - ▶ GR goal not satisfied
 - ▶ Not complete, verifiable, modifiable or traceable
 - ▶ Coordinate system information missing
 - ▶ Ambiguous rank function
 - ▶ Inputs not checked in code
 - ▶ User not informed of their responsibility to use tool with correct statistical model

Concluding Remarks

- Hopefully motivated assurance cases for SC
- Quality is improved by looking at a problem from different perspectives, assurance cases provide a systematic and rigorous way to introduce a new perspective
- An assurance cases will likely use the same documentation and ideas used in CAS 741
- However, an assurance case can focus and direct efforts right from the start of the project

References I



W. Spencer Smith, Mojdeh Sayari Nejad, and Alan Wassying.

Assurance cases for scientific computing software (poster).

In ICSE 2018 Proceedings of the 40th International Conference on Software Engineering, May 2018.

2 pp.



W. Spencer Smith, Mojdeh Sayari Nejad, and Alan Wassying.

Raising the bar: Assurance cases for scientific computing software.

Computing in Science and Engineering, 23(1):47–57, February 2020.