

# CAS 741, CES 741 (Development of Scientific Computing Software)

Fall 2018

## 22 Assurance Case

Dr. Spencer Smith

Faculty of Engineering, McMaster University

November 21, 2018



# Assurance Case

- Administrative details
- Feedback on MG
- Final documentation
- Questions?
- License and copyright
- Assurance cases

# Administrative Details

- Course evaluation
  - ▶ Thurs, Nov 22, 10:00 am to Thurs, Dec 6, 11:59 pm
  - ▶ <https://evals.mcmaster.ca>
- No class on Friday, Nov 30
- GitHub issues for colleagues
  - ▶ Assigned 1 colleague (see Repos.xlsx in repo)
  - ▶ Provide at least 5 issues on their MIS
  - ▶ Grading as before
  - ▶ Due by Tuesday, Dec 5, 11:59 pm
- Today is the last “lecture”
- Next week for presentations
- Following Tuesday for Discussion

# Administrative Details: Deadlines

MIS	Week 11	Nov 23
Unit VnV or Impl. Present	Week 12, 13	Wed, Nov 28, Dec 5
Unit VnV Plan	Week 13	Dec 3
Final Doc	Week 14	Dec 10

# Administrative Details: Presentation Schedule

- Unit VnV Plan or Impl. Present
  - ▶ Wednesday (Nov 28): Brooks, Vajiheh
  - ▶ Wednesday (Dec 5): Olu, Karol
- Can present anything related to the implementation or testing
  - ▶ Code
  - ▶ Tools used
  - ▶ Testing
  - ▶ As always it is fine to show work in progress
  - ▶ Good to bring questions to the class

# Feedback on MG

- Good work! Good starting point. May have to modify design as a response to MIS
- If multiple anticipated changes map to the same module, you should explain why
- If one anticipated change maps to multiple modules, you should explain why
- You don't have an input parameters module

# 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
  - ▶ Specific explanation in Revision History
  - ▶ Comments in tex file

# Final Documentation

- Requirements Document revised and improved
- Design Documents revised and improved
- Test Plan revised and improved
- Test Report
- Source Code



# 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
- 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
- Traceability to modules in module guide

# Coding Style

- Having a coding standard is more important than which standard you use
- Examples
  - ▶ Google guides
    - ▶ Python
    - ▶ C++
    - ▶ Java
  - ▶ Mozilla Developer Network
  - ▶ NASA C Style Guide
- Your decisions on style may evolve over the project
- Important to be consistent

# 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

# Final Documentation

- Traceability between documents
- Look for an obvious requirement to see if it is in the requirements document and traceable through the other documents
- 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

# Final Documentation: Test Plan

- See template

# Final Documentation: Test 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.

# Test 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

# Test 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



# Test 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 Test Report Documents

- Screenholders
- 2D Physics Based Game (Uses doxygen)
- 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

# Questions?

- Questions about MIS documentation?
- Questions about implementation presentations?

# 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

# GNU Questions

- Question 1
  - ▶ You modify some Linux source files to install Linux on your embedded device
  - ▶ You write software to run on this new Linux “box”
  - ▶ What software falls under the GPL?
  - ▶ [Answer](#)
- Question 2
  - ▶ You want to distribute object code compiled by gcc, where gcc is under GPL
  - ▶ Is your object code under GPL?
  - ▶ [Answer](#)

# 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](#)

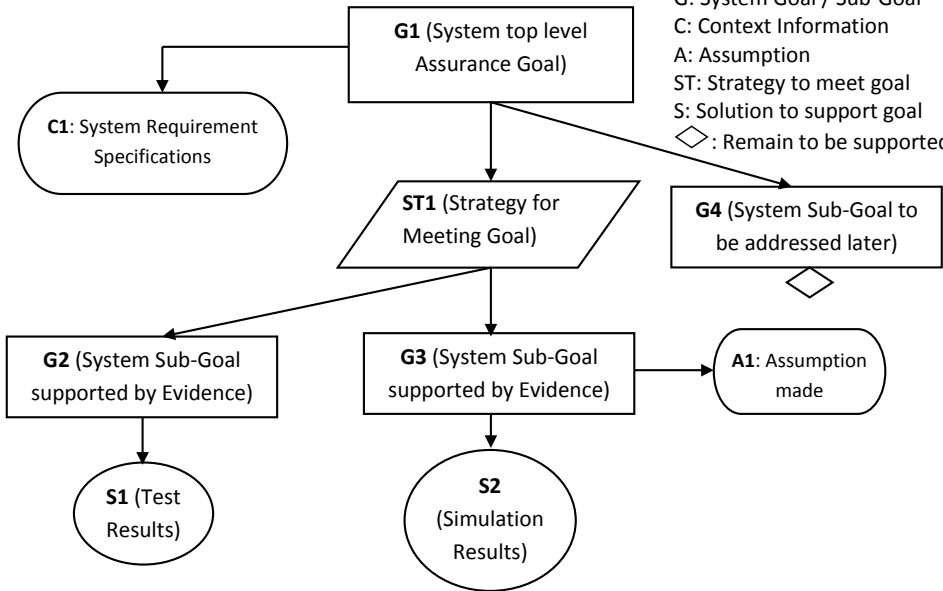
# Assurance Cases in Scientific Computing [1]

- 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

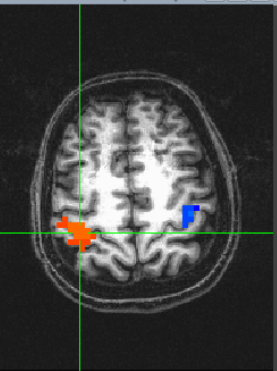
Coronal

◆ Original View  
 ◆ AC-PC Aligned  
 ◆ Talairach View

☒ See Markers

☒ See OverLay

[A] AFNI: tmp/LRtap/m



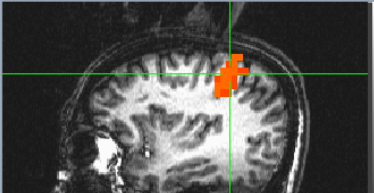
Colr  
 Swap  
 Norm

c    
 b    
 r    
 g    
 i    
 9    
 z    
 pan  
 crop

141

left=Right Float [2%-98%]

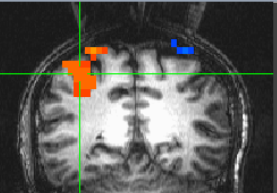
[A] AFNI: tmp/LRtap/mdef3d\_01+ori



Colr  
 Swap  
 Norm

c    
 b    
 r    
 g    
 i

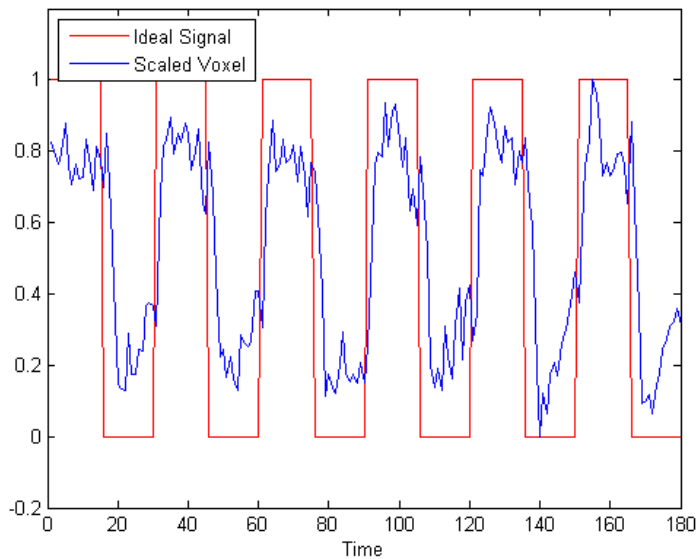
[A] AFNI: tmp/LRtap/m

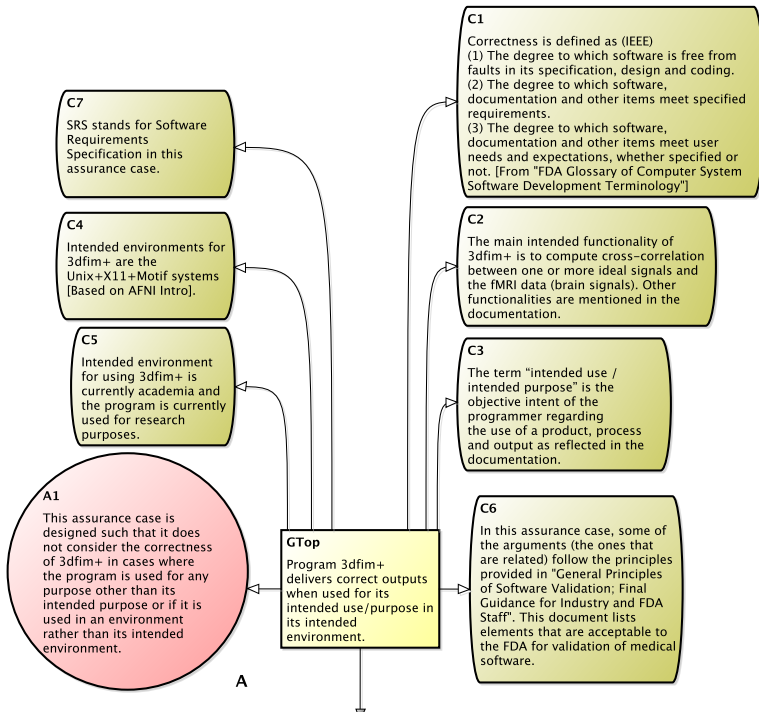


Colr  
 Swap  
 Norm

c    
 b    
 r    
 g    
 i

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





### G<sub>Top</sub>

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

### S<sub>Top</sub>

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<sub>Top</sub> is valid.

### J<sub>Top</sub>

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

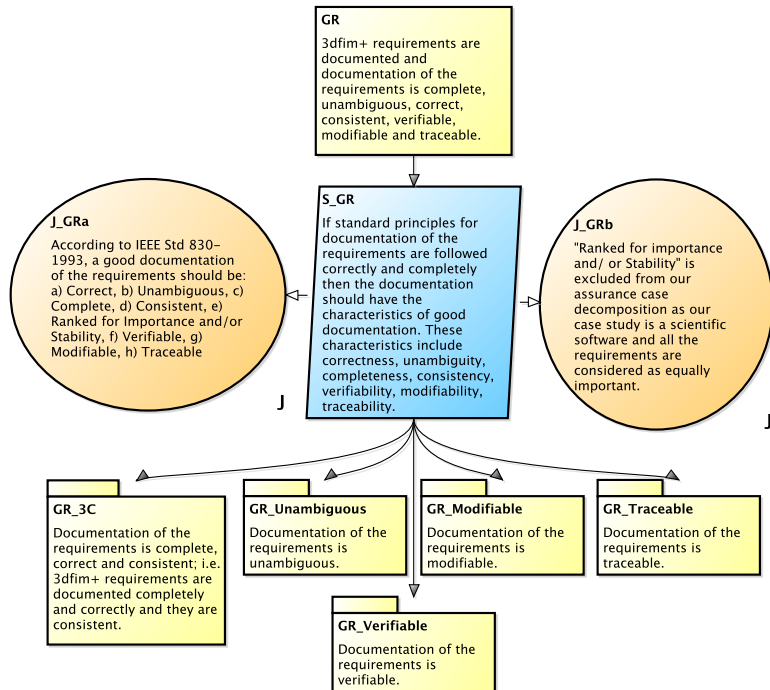
The design of 3dfim+ complies with its requirements and it is complete, unambiguous, correct, consistent, verifiable, modifiable and traceable.

### 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.

#### 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.

#### Modifiable.2

There is no duplication between the requirements.

#### 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.

#### Modifiable.3

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

#### 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\_ModifiableC

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

**C\_ModifiableB**  
List of the team members.

#### E\_Modifiable.1

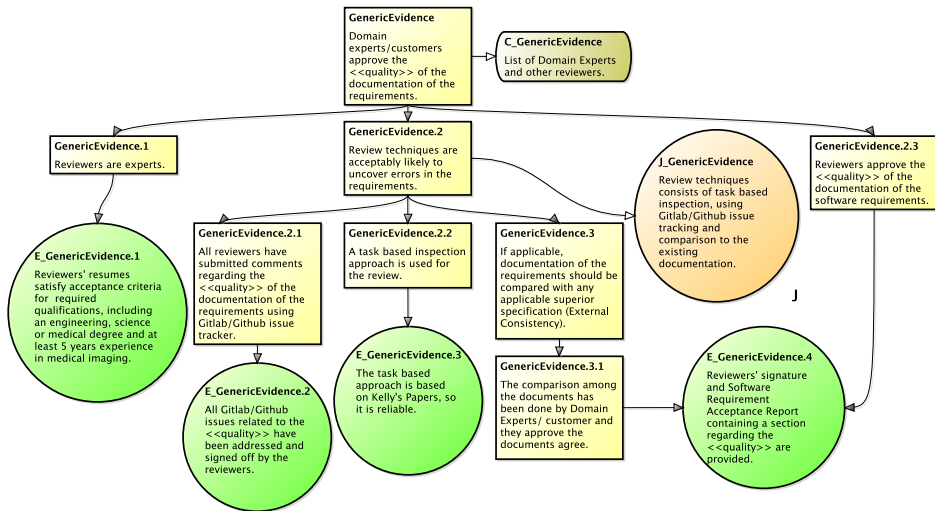
The standard template.

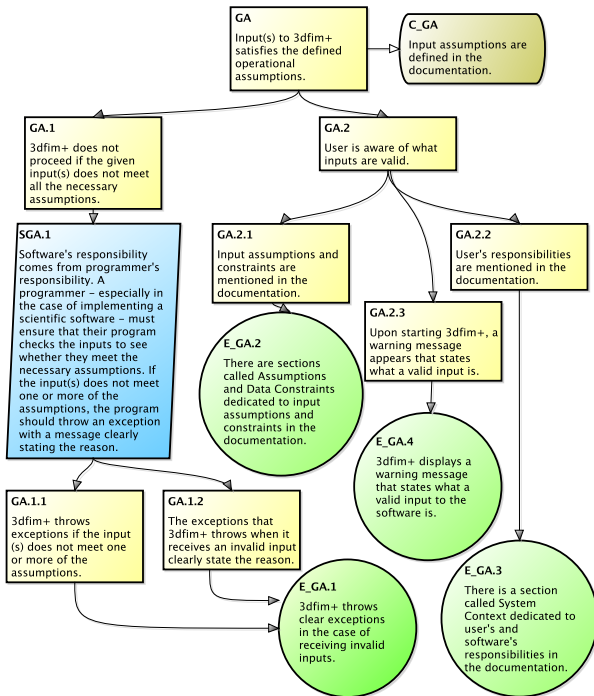
#### E\_Modifiable.2

Team members' resumes.

#### GenericEvidence

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







# 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 Wassyng.

Assurance cases for scientific computing software.

Prepared for ICSE Submission, November 2017.