Module Guide for Slope Stability Analysis Program

Henry Frankis and Brooks MacLachlan

November 5, 2018

1 Revision History

Date	Version	Notes
31/10/2018	1.0	Initial template and module updates
11/01/2018	1.1	Anticipated changes updates

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

Symbol	Description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
SSP	Slope Stability Analysis Program
UC	Unlikely Change

[This section is not on the template but I think that it should be since this document introduces many new acronyms —BM]

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). In the best practices for Scientific Computing (SC), Wilson et al. (2013) advise a modular design, but are silent on the criteria to use to decompose the software into modules. We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

[There is an item here in the template saying "each data structure is used in only one module", which just seems wrong to me. Maybe each data structure is implemented by a dedicated module, but then many other modules can still use that data structure (module). —BM]

In a rational design process, the Module Guide (MG) is developed after completing the Software Requirements Specification (SRS) (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module hierarchy that was constructed according to the likely changes. Section 6 specifies the connections

between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to Slope Stability analysis Program (SSP). According to the likeliness of the change, the possible changes are classified into two categories. Each Anticipated Change (AC) is listed in Section 4.1, and each Unlikely Change (UC) is listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The algorithm for the overall operation procedure of the program.

AC3: The format of the initial input data.

AC4: The constraints on the initial input data.

AC5: The format of the final output data.

AC6: The constraints on the factor of safety and interslice forces.

AC7: The algorithm for determining the critical slip surface.

AC8: The criteria for a slip surface to be considered physically realistic.

AC9: The weighting scheme for comparing slip surface's factors of safety.

AC10: The algorithm for dividing a slip surface into vertical slices.

AC11: The algorithm used to calculate the factor of safety for a slip surface.

AC12: The algorithm for calculating soil properties of and forces on individual slices.

AC13: The implementation of the array data structure.

AC14: The method of generating pseudo-random numbers.

AC15: The method for creating plots.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or keyboard, Output: File, memory, and/or screen).

UC2: The goals of SSP, to determine a slope's critical slip surface and corresponding factor of safety, will not change.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below are the modules that will actually be implemented. Modules are numbered by **M** followed by a number.

M1: Hardware-Hiding

M2: Control

M3: Input

M4: Output

M5: Genetic Algorithm

M6: Kinematic Admissibility

M7: Slip Weighting

M8: Slip Slicing

M9: Morgenstern-Price Calculation

M10: Slice Property Calculation

M11: Array Data Structure

M12: Random Number Generation

M13: Plotting

Note that M1 is a commonly used module and is already implemented by the Operating System (OS).

Level 1	Level 2
Hardware-Hiding	
Behaviour-Hiding	Control
	Input
	Output
	Genetic Algorithm
	Kinematic Admissibility
	Slip Weighting
	Slip Slicing
	Morgenstern-Price Calculation
	Slice Property Calculation
Software Decision	Array Data Structure
	Random Number Generation
	Plotting

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

[This section seems to cover the same topic as the traceability matrix in Section 8. I am treating it as a place where I can provide more detailed justification for the design than can be found in a traceability matrix. Was that the intention for this section? My second paragraph here, in particular, might be out of place since it is related to anticipated changes and not requirements? Not sure where else to put it though. Maybe within Section 8 itself? —BM]

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2. The program uses the Morgenstern-Price method to calculate the factor of safety, which is implemented by M9. Further information on the Morgenstern-Price algorithm can be found in Zhu et al. (19 February 2005). A genetic algorithm is used to generate potential slip surfaces to compare to find the critical slip surface, implemented in M5. Further information on use of a genetic algorithm in relation to a slope stability problem can be found in Li et al. (25 June 2010).

While the division of a slip surface into vertical slices and the calculation of properties of each slice are part of the Morgenstern-Price method, they have been separated into their own modules, M8 and M10. This is due to AC11, as these modules would be common between different methods for calculating the factor of safety.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. If the entry is *SSP*, the module will be implemented by SSP. Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

7.1 Hardware-Hiding (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding

Secrets: The contents of the required behaviors.

Services: Includes programs that provide externally visible behaviors of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Control (M2)

Secrets: The algorithm for coordinating how the program runs.

Services: Drives the program through the required tasks.

Implemented By: SSP

7.2.2 Input (M3)

Secrets: The format and structure of the input data.

Services: Reads, verifies, and stores the input data. This includes a set of coordinates for each slope layer, soil properties for each layer, specifically effective angle of friction, effective cohesion, dry unit weight, and saturated unit weight, and the coordinates and unit weight of water for a water table, if one exists. This also includes an expected range for the start and end points of the critical slip surface and a decision on whether to use a half-sine or the constant 1 for the interslice normal to shear force ratio variability function.

Implemented By: SSP

7.2.3 Output (M4)

Secrets: The format and structure of the output data.

Services: Outputs the results of the calculations, including the factor of safety for the critical slip surface and a plot of the critical slip surface on the slope geometry. Verifies that the output meets physical constraints. Also outputs input values.

Implemented By: SSP

7.2.4 Genetic Algorithm (M5)

Secrets: Algorithm for identifying the critical slip surface with the minimum factor of safety.

Services: Generates a set of potential slip surfaces and searches the set for the critical slip surface with the minimum factor of safety.

Implemented By: SSP

7.2.5 Kinematic Admissibility (M6)

Secrets: The kinematic admissibility criteria for determining if a given slip surface is realistic physically.

Services: Tests slip surfaces for physically unlikely features including sharp angles, points outside of the soil bed, or downward concavity.

Implemented By: SSP

7.2.6 Slip Weighting (M7)

Secrets: The criteria for weighting each slip surface in a given set of slip surfaces based on how closely they resemble the critical slip surface.

Services: Assigns weights to each slip surface in a given set of slip surfaces based on their factors of safety. A slip surface with a lower factor of safety will have a higher weight as it is closer to the critical slip surface.

Implemented By: SSP

7.2.7 Slip Slicing (M8)

Secrets: Algorithm to determine the coordinates of where the slip surface interslice nodes occur.

Services: When preparing a slip surface for analysis by the Morgenstern-Price Module (M9) the x-coordinates defining the boundaries of the slices are identified and stored in a vector.

Implemented By: SSP

7.2.8 Morgenstern-Price Calculation (M9)

Secrets: The algorithm for calculating the factor of safety of a given slip surface.

Services: Calculates the factor of safety of a given slip surface, through implementation of a Morgenstern-Price slope stability analysis method.

Implemented By: SSP

7.2.9 Slice Property Calculation (M10)

Secrets: Algorithm to calculate slice properties, such as forces and soil properties, based on the location of the slice with respect to the different soil layers.

Services: Calculates the weight, hydrostatic forces, interslice water forces, and material properties of each slice. Identifies which soil layer the slice is in to assign properties from that soil layer, and uses a weighting scheme when the slice crosses multiple soil layers.

Implemented By: SSP

7.3 Software Decision

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structures and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Array Data Structure (M11)

Secrets: The data structure for an array data type.

Services: Provides array manipulation, including building an array, accessing a specific entry, slicing an array etc.

Implemented By: Matlab

7.3.2 Random Number Generation (M12)

Secrets: Algorithm for providing pseudo-random numbers between 0 and 1.

Services: Randomly produces numbers between 0 and 1, using a chaotic function with an external seed.

Implemented By: Matlab

7.3.3 Plotting (M13)

Secrets: The data structures and algorithms for plotting data graphically.

Services: Provides functions for plotting data.

Implemented By: Matlab

8 Traceability Matrix

This section shows two traceability matrices: Table 2 shows which modules cover each Requirement (R) and Table 3 shows how modules map to the anticipated changes. There are two cases where a single module covers multiple anticipated changes. AC3 and AC4 are both covered by M3, and AC5 and AC6 are both covered by M4. In these cases, the anticipated changes are related to two services that should always happen in tandem. If input is being read, it should always be verified. If output is being delivered, it should always be verified. Thus, it makes sense to combine the two services into a single module.

Requirement	Modules
R1	M1 M2 M3 M11
R2	M2 M3 M11
R3	M2 M5 M11 M12
R4	M8 M9 M10 M11
R5	M5 M7 M11
R6	M2 M6 M4 M11
R7	M1 M2 M4 M11 M13
R8	M1 M2 M4 M11
R9	M1 M2 M4 M11 M13
R10	M1 M4 M11 M13

Table 2: Trace Between Requirements and Modules

Anticipated Change	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M3
AC5	M4
AC6	M4
AC7	M5
AC8	M6
AC9	M7
AC10	M8
AC11	M9
AC12	M10
AC13	M11
AC14	M12
AC15	M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use hierarchy between the modules. The graph is a Directed Acyclic Graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

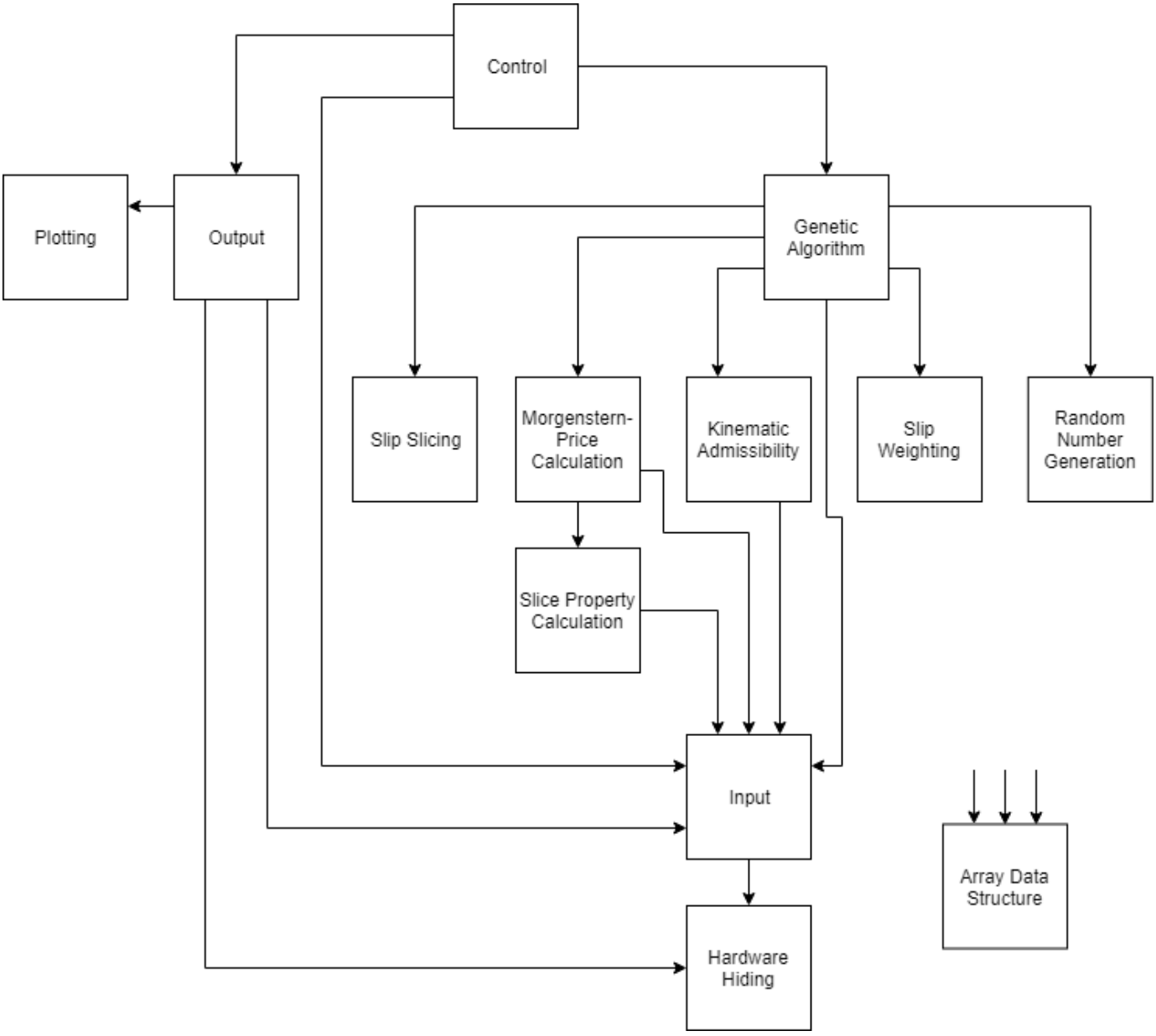

Figure 1: Use Hierarchy Diagram

10 Bibliography

Yu-Chao Li, Yun-Min Chen, Tony L.T Zhan, Dao-Sheng Ling, and Peter John Cleall. An efficient approach for locating the critical slip surface in slope stability analyses using a real-coded genetic algorithm. *Can. Geotech. J.*, (47):806–820, 25 June 2010.

D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 408–417, Piscataway, NJ, USA, 1984. IEEE Press. ISBN 0-8186-0528-6.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM, vol. 15, no. 2, pp. 1053-1058*, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

Greg Wilson, D.A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H.D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumblet, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *CoRR*, abs/1210.0530, 2013.

D.Y. Zhu, C.F. Lee, Q.H. Qian, and G.R. Chen. A concise algorithm for computing the factor of safety using the morgenstern–price method. *Can. Geotech. J.*, (42):272–278, 19 February 2005.