Module Guide: Kaplan [include software's name —SS]

Jen Garner

November 5, 2018

1 Revision History

Date	Version	Notes
November 5th, 2018 (Monday)	1.0	First draft

Contents

List of Tables

List of Figures

2 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). 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 layed out by Parnas et al. (1984), as follows:

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

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (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 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 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 8 describes the use relation between modules.

3 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 3.1, and unlikely changes are listed in Section 3.2.

3.1 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: Hardware used to run the software.

AC2: Format of the input data.

AC3: RMSD module may be rewritten as a behaviour hiding module.

AC4: The ring may change to another data structure.

AC5: The calculation for Fit_G may be expanded to include more formats than initially described in the SRS document.

AC6: Kaplan may be incorporated into another software package, thus making it a library not a standalone program.

AC7: As per AC6, Kaplan may be required to run on cell phones through a web interface. The input/output device therefore will change from a keyboard and mouse to a touchscreen.

AC8: The programming language may change to improve performance.

AC9: The external program used to run the energy calculations may change or multiple choices may become available (depending on user input).

3.2 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: The genetic algorithm component is unlikely to be exchanged for another algorithm.

UC2: The energy is unlikely to be calculated using methods other than those from quantum mechanics.

UC3: The goal of Kaplan is unlikely to change from locating conformers.

4 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, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding

M2: GA Input

M3: GA Control

M4: Fit_G

M5: Tournament

M6: Crossover & Mutation

M7: Ring

M8: Output

M9: Molecule Input

M10: Energies

M11: RMSD

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	GA Input GA Control Fit_G Tournament Crossover & Mutation Ring Output
Software Decision Module	Molecule Input Energies RMSD

Table 1: Module Hierarchy

5 Connection Between Requirements and Design

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.

6 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. Also indicate if the module will be implemented specifically for the software.

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.

6.1 Hardware Hiding Modules (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

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour 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: –

6.2.1 GA Input Module (M2)

Secrets: The format and structure of the input data for the genetic algorithm, including how the input data is verified.

Services: Communicates the size of the Ring data structure, the number of mating events to perform, the maximum number of mutations, the maximum number of swaps, the number of conformers in the optimization, the size of the initial population, and which fitness function to use to M3. Ensures all input data is of the correct type and within reasonable bounds. These values are static and should not change during the optimization.

Implemented By: Kaplan

6.2.2 GA Control Module (M3)

Secrets: How the pieces of the genetic algorithm work together to find and optimize conformer geometries.

Services: Initializes the Ring data structure and calls the Tournament to update the Ring. When the number of mating events specified by the user has been completed, calls the Output module. If the Ring has empty entries for energy at the end of the program's execution, then this module can call M4 to fill in the missing values.

Implemented By: Kaplan

6.2.3 Fit_G Module (M4)

Secrets: How to calculate the fitness of a set of conformers.

Services: Calculate the fitness of given population members from the Ring.

Implemented By: Kaplan

6.2.4 Tournament Module (M5)

Secrets: How to conduct Tournament selection for the genetic algorithm.

Services: Chooses participants, winners, and losers in the tournament such that these population members can be updated in the Ring.

Implemented By: Kaplan

6.2.5 Crossover & Mutation Module (M6)

Secrets: How to generate new population members.

Services: Generates new population members to put in the Ring data structure.

Implemented By: Kaplan

6.2.6 Ring Module (M7)

Secrets: The data structure for the Ring.

Services: Defines the potential solutions to the conformer optimization in the form of dihedral angles (and energies if calculated). Determines if a new population member can be added, and where they are added.

Implemented By: Kaplan

6.2.7 Output Module (M8)

Secrets: The format and content of the output data.

Services: Provides the results of the conformer optimization to the user. Receives the input geometry from M3 to reconstruct the new conformer geometries.

Implemented By: Kaplan

6.3 Software Decision Module

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 structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

6.3.1 Molecule Input Module (M9)

Secrets: The format and structure of the input data for the molecule, including how the input data is verified.

Services: Converts the input data between formats, extracts dihedral angles, reconstructs geometries from dihedral angles and initial bond angles and lengths.

Implemented By: Vetee, Openbabel O'Boyle et al. (2011) oba (2018)

6.3.2 Energies Module (M10)

Secrets: How to calculate the energy of a conformer geometry.

Services: Calculates the energy for a given geometry based on user-specified basis set and method.

Implemented By: Psi4 Parrish et al. (2017), Gaussian Frisch et al. (2016), Horton Verstraelen et al. (2017)

6.3.3 RMSD Module (M11)

Secrets: How to calculate the root-mean-square deviation for a set of conformers.

Services: Calculates the RMSD for a set of geometries.

Implemented By: rmsd (charnley)

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1	M9
R2	M4, M7
R3	M4, M11, M10
R4	M4, M10
R5	M8, M9
NFR1	M4
NFR2	M2-M11, but especially M3
NFR3	M2-M11
NFR4	M9, M2
NFR5	M10, M11, M9, M8

Table 2: Trace between requirements and modules.

As in Table 3, there are two modules that change as a result of AC2. The input will be divided into two modules - one module (M2) is responsible for handling the genetic algorithm inputs and the other module (M9) is responsible for handling the molecular inputs. The author has already written a software package that can handle molecular geometry (and its conversion to other formats). This package also interfaces with Openbabel O'Boyle et al. (2011). The author has decided to use a Behaviour-Hiding module for the genetic algorithm, as these algorithms are very dependent on the format of the data structure used to represent the problem (and thus converting to a format acceptable to another external program might change the results significantly). Furthermore, one of the goals of the program is to make the package work on high-performance computing clusters. If there are fewer external programs necessary to run Kaplan, then it will be easier to install the program. All modules will change upon change of the programming language, hence the reason AC8 is linked to all Behaviour-Hiding and Software Decision modules.

AC	**Modules**
AC1	M1
AC2	M9 M2
AC3	M11
AC4	M7
AC5	M4
AC6	M3
AC7	M8
AC8	M2-M11
AC9	M10

Table 3: Trace between anticipated changes and modules.

8 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 relation between the modules. It can be seen that 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.

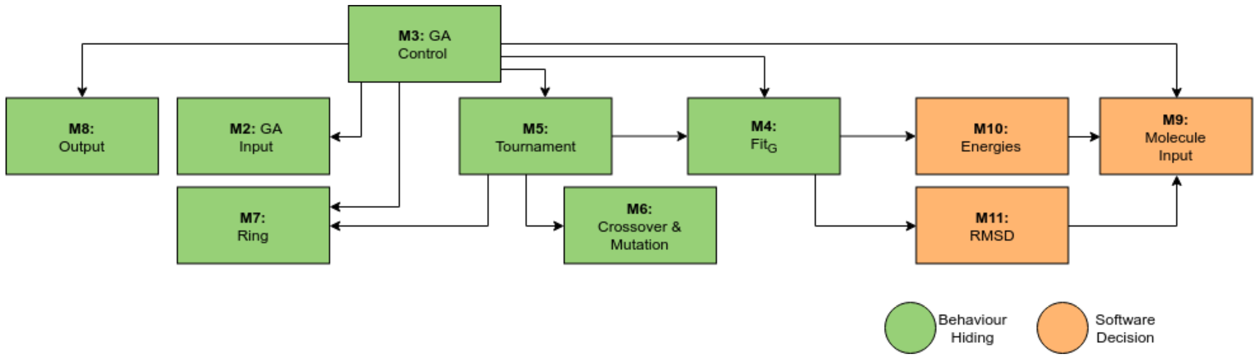

Figure 1: Use hierarchy among modules. Note the hardware-hiding module has been omitted for clarity.

References

The open babel package, version 2.3.1, 2018. URL http://openbabel.org.

Jimmy Charnley Kromann (charnley). rmsd, 2018. URL https://github.com/charnley/rmsd.

M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox. Gaussian16 Revision B.01, 2016. Gaussian Inc. Wallingford CT.

N M O'Boyle, M. Banck, James C. A, C. Morley, T. Vandermeersch, and G. R. Hutchison. Open babel: An open chemical toolbox. *J. Cheminf.*, 3, 2011. doi: 10.1186/1758-2946-3-33.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):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.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince III, E. G. Hohenstein, U. Bozkaya, A. Yu. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer III, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill. Psi4 1.1: An open-source electronic structure program emphasizing automation, advanced libraries, and interoperability. *J. Chem. Theory Comput.*, 13:3185–3197, 2017.

Toon Verstraelen, Pawel Tecmer, Farnaz Heidar-Zadeh, Cristina E. González-Espinoza, Matthew Chan, Taewon D. Kim, Katharina Boguslawski, Stijn Fias, Steven Vandenbrande, Diego Berrocal, and Paul W. Ayers. Horton 2.1.0, 2017. URL http://theochem.github.com/horton/.