The Modular Structure of Complex Systems

David L. Parnas, Paul C. Clement, and David M. Weiss
ICSE 1984

Presented by:
Seyed Masoud Sadjadi and Wei Zhu
CSB70 Advanced Software Engineering
(Cheng, 2002)

Main Paper Overview

Motivation:
Software systems are getting more complex.

Problem:
- So many design decisions and implementation details.
- Completeness of software.
- Duplication.
- Finding modules related to a change.

Goal:
- Keep a complex system under intellectual control
- Keep maintenance cost down (also development and deployment costs).

Solution:
- Dividing complex systems into modules.
- Using information hiding to define modules.
- Using hierarchical structure to classify modules.

Main Paper Technical Overview

Design through documentation:

Module Guide:
- Helps developers and maintainers of the system to find the affected module by a change.

Module Interface Document:
- Contains precise and complete description of interface to each module.

Module Design Document:
- An internal design document for each implementation of a module.

Requirements Document:
- Module guide refers to it for the related changes.
Main Paper Impacts

- Big impact on software engineering community:
  - Information Hiding.
  - Abstract Data Type.
  - Separation of Concerns.
  - Object Orientation.
  - Component-base Software.

  All the same principle.

First Related Paper Overview

Towards a Semantically Enhanced Component Trader Architecture.

- Motivation:
  - Software development is moving toward component-based design.
- Goal:
  - Providing a large-scale open systems trading.
- Problem:
  - Appearance-base (interface) definition is not sufficient to find a needed component.
- Solution:
  - Adding behavior-based (semantic) specification of component to appearance-based definition.
  - Developing a "semantically enhanced" component trader.

First Related Paper Technical Overview

- Background:
  - Name Service (e.g. DNS)
  - Directory Service (e.g. White Pages)
  - Trading Service (e.g. Yellow Pages)
  - No name is required to find a component.
  - Just the developers/components need (specification of service type).
- Component Description Language (CDL):
  - Like IDL in CORBA.
  - Start from CDL to do component matching process.
- Encoding component behavior:
  - Text-based: no sufficient info for classification.
  - Lexical descriptor-based: Vocabulary by subject experts.
  - Specification-based: Formal is good.
- Conceptual structures & Knowledge representation.
First Related Paper (cont.)

Why this paper extends Parnas' work?

- A module guide helps a developer or a maintainer to find a module for change or other reasons.
- A trader finds components that matches the needed component.
- The hierarchical design of component in the trader is an extension to Parnas' hierarchical module structure that has different application.
- The ideas for the hierarchically classifying components has originated from Parnas' module guide.

Second Related Paper Overview

Coping With Software Change Using Information Transparency.

- Motivation:
  - Changes in cross-cutting concerns are expensive.
- Goal:
  - Localizing cross-cutting concerns.
- Standard Solution:
  - Using AGP to localize these concerns.
- Problem with the Standard Solution:
  - You should anticipate the future changes.
  - Hard to define and factorize and also coarse grain.
- Solution Proposed in This Paper:
  - Information Transparency:
    - Using tools to economically identify the related code.
    - Naming conventions, formatting style, and so on.

Second Related Paper (cont.)

How this paper extends Parnas' work?

- Information hiding tries to localize some of the concerns as secrets of a module.
- Hierarchical module guide localize related modules in a tree.
- Information transparency is an extension to information hiding and hierarchical module design:
  - Information transparency localizes the cross-cutting aspects among modules that are not localized in the module guide.
  - Module guide uses references to requirement specifications, which is too time consuming, inconvenient, and expensive.
- Information transparency is a complementary to information hiding.
Third Related Paper Overview

Assuring Good Style for Object-Oriented Programs.

- Motivation:
  - How to write a software as modular as possible?
- Goal:
  - Writing objects (modules) with minimum dependencies to other objects (modules).
- Problem:
  - Lack of good programming style principles for modularity.
  - Lack of principles to write programs for change.
- Solution in this paper:
  - The law of Demeter

Third Related Paper Technical Overview

- The law of Demeter:
  - Rules for structure of classes.
  - Rules for how methods are written:
    - Restricting the message-sending structure of methods.
  - Each method can send message to only:
    - Argument objects.
    - Self pseudo-variables.
    - The immediate subparts of self (this).

Third Related Paper (cont.)

- How this paper extends Pamas’ work?
  - The law of Demeter extends information hiding by introducing structure hiding.
  - It reduces the cost of software changes in complex systems by extending restricted modules (subject to change) to information restriction.
  - Localizing “type” information
    - A method should be aware of just the very closely related classes, and is ignorant of the rest of the system (reducing programming complexity).
  - A complementary to Pamas’ work.
Uncited Paper Overview

Designing Reusable Classes

Motivation:
- A major motivation for OO programming is software reuse.
- How to design reusable classes to reduce development time and maintenance cost?

Goal:
- Program components must be designed for reusability.
- Describes and organizes techniques that are used to design more usable software.

Solution:
- Polymorphism, inheritance, framework, and some rules of thumb for developing standard protocols, for finding abstract classes, and for finding frameworks.

Uncited Paper Technical Overview

- Major attributes of OO languages
  - Data abstraction
  - Inheritance
  - Polymorphism
- Framework, toolkit...we've known this from class
- How does OO improve software reusability
  - Class definition provides modularity and information hiding
  - Late-binding of procedure calls means objects require less information about each other
  - A polymorphic procedure is easier to reuse than non-polymorphic one
  - Inheritance enables a class to be reused and extended by making subclasses

Uncited Paper (cont.)

- Why this paper should have cited Parnas’ paper?
  - OO features that provide reusability are enabled by information hiding.
  - Encapsulation, polymorphism, inheritance
  - "Object-oriented" is just for reinventing the ideas behind the older concepts such as "abstract data type", "information hiding", and "modularization"
  - "Information hiding" is becoming a general rule in software design, a general rule in people’s mind...
## References