SFWR ENG 3A04: Software Design II

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Term 1, 2008–2009

Acknowledgments: Material based on *Software Architecture Design* by Tao et al. (Chapter 4)

Outline of Part I

1. OO Analysis and Design
    - OO Analysis
    - OO Design
2. Questions???

Outline of Part II

3. Overview
4. Principle of Low Coupling and High Cohesion
5. Open-Closed Principle
6. Liskov substitution principle
7. Dependency Inversion Principle
8. Law of Demeter
9. Other Design Principles for Security
   - Principle of Least Privilege
   - Principle of Fail-Safe Defaults
   - Principle of Economy of Mechanism
   - Principle of Complete Mediation
   - Principle of Open Design
   - Principle of Separation of Privilege
   - Principle of Least Common Mechanism
   - Principle of Psychological Acceptability

Part I

Review of Previous Lecture

Part II

Today's Lecture

Overview

- A design process is not to simply identify one possible solution for a problem and then furnish the details of it

General Design Principles Overview

- A design process is not to simply identify one possible solution for a problem and then furnish the details of it
- A good designer has to identify several alternative designs for a problem

General Design Principles Overview

- A design process is not to simply identify one possible solution for a problem and then furnish the details of it
- A good designer has to identify several alternative designs for a problem
- In the selection process, the designer is guided by design principles

General Design Principles Overview

- A design process is not to simply identify one possible solution for a problem and then furnish the details of it
- A good designer has to identify several alternative designs for a problem
- In the selection process, the designer is guided by design principles
- These principles build on the ideas of simplicity and restriction

General Design Principles Overview

- A design process is not to simply identify one possible solution for a problem and then furnish the details of it
- A good designer has to identify several alternative designs for a problem
- In the selection process, the designer is guided by design principles
- These principles build on the ideas of simplicity and restriction
- Simplicity makes the proposed solutions easy to understand (Less can go wrong with simple designs)

General Design Principles Principle of Low Coupling and High Cohesion

In general:

- Cohesion within a module is the degree to which communication takes place among the module's elements

General Design Principles Principle of Low Coupling and High Cohesion

In general:

- Cohesion within a module is the degree to which communication takes place among the module's elements

- Coupling describes the degree to which modules depend directly on other modules

General Design Principles Principle of Low Coupling and High Cohesion

In general:

- Cohesion within a module is the degree to which communication takes place among the module's elements
- Coupling describes the degree to which modules depend directly on other modules
- Effective modularization is accomplished by maximizing cohesion and minimizing coupling

General Design Principles Principle of Low Coupling and High Cohesion

In general:

- Cohesion within a module is the degree to which communication takes place among the module's elements
- Coupling describes the degree to which modules depend directly on other modules
- Effective modularization is accomplished by maximizing cohesion and minimizing coupling
- This principle helps to decompose complex tasks into simpler ones

General Design Principles

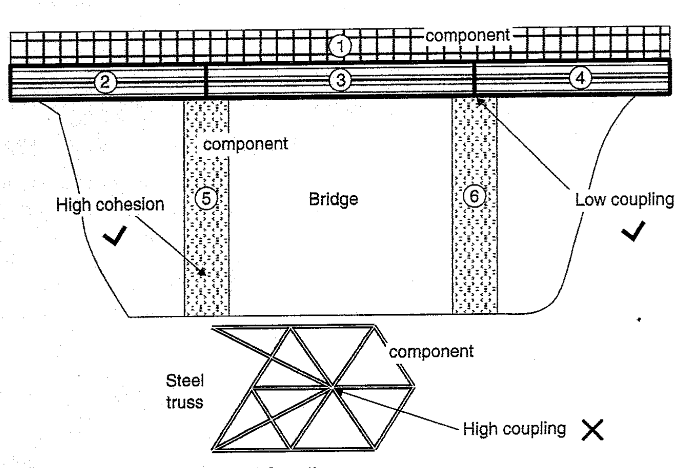


Figure: Cohesion and Coupling

General Design Principles Principle of Low Coupling and High Cohesion

In the comtext of OO Design:

- A system with highly inter-dependable classes is very hard to maintain

General Design Principles Principle of Low Coupling and High Cohesion

In the comtext of OO Design:

- A system with highly inter-dependable classes is very hard to maintain

- A change in one class may result in cascading updates of other classes

General Design Principles Principle of Low Coupling and High Cohesion

In the comtext of OO Design:

- A system with highly inter-dependable classes is very hard to maintain
- A change in one class may result in cascading updates of other classes
- We should avoid tight-coupling of classes (Identified using analysis class diagram)

General Design Principles Principle of Low Coupling and High Cohesion

In the comtext of OO Design:

- A system with highly inter-dependable classes is very hard to maintain
- A change in one class may result in cascading updates of other classes
- We should avoid tight-coupling of classes (Identified using analysis class diagram)
- A pair of classes which has dependency association on each other is called tightly-coupled

General Design Principles Principle of Low Coupling and High Cohesion

In the comtext of OO Design:

- A system with highly inter-dependable classes is very hard to maintain
- A change in one class may result in cascading updates of other classes
- We should avoid tight-coupling of classes (Identified using analysis class diagram)
- A pair of classes which has dependency association on each other is called tightly-coupled
- Tight coupling might be removed by introducing new classes or inheritance

General Design Principles Overview

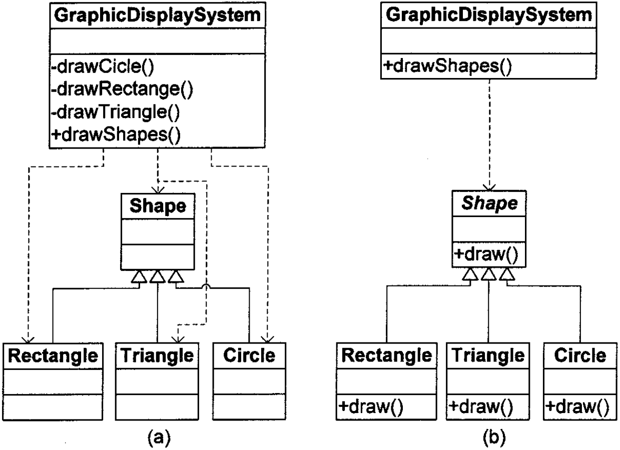


Figure: Vertical override operation (Used for decoupling)

General Design Principles Principle of Low Coupling and High Cohesion

We should seek:

- Less inter-dependency

General Design Principles Principle of Low Coupling and High Cohesion

We should seek:

- Less inter-dependency

- Easy expansion

General Design Principles Principle of Low Coupling and High Cohesion

We should seek:

- Less inter-dependency
- Easy expansion
- Simplicity and elegancy in implementation

$$\text{good design} \implies \text{simple} \quad \wedge \quad \text{elegant}$$

General Design Principles Principle of Low Coupling and High Cohesion

We should seek:

- Less inter-dependency
- Easy expansion
- Simplicity and elegancy in implementation

$$\text{good design} \implies \text{simple} \quad \wedge \quad \text{elegant}$$

is equivalent to

$$\neg\text{simple} \quad \vee \neg\text{elegant} \implies \neg\text{good design}$$

General Design Principles Principle of Low Coupling and High Cohesion

- A cohesive class is one that performs a set of closely related operations

General Design Principles Principle of Low Coupling and High Cohesion

- A cohesive class is one that performs a set of closely related operations
- If a class performs more than one non-related functions, it is said to be lack of cohesion

General Design Principles Principle of Low Coupling and High Cohesion

- A cohesive class is one that performs a set of closely related operations
- If a class performs more than one non-related functions, it is said to be lack of cohesion
- A lack of cohesion makes the overall structure of the software hard to manage, expand, maintain, and modify

Principle of Low Coupling and High Cohesion

- A cohesive class is one that performs a set of closely related operations
- If a class performs more than one non-related functions, it is said to be lack of cohesion
- A lack of cohesion makes the overall structure of the software hard to manage, expand, maintain, and modify
- By improving information hiding you will generally be improving the coupling and cohesion

Principle of Low Coupling and High Cohesion

- A cohesive class is one that performs a set of closely related operations
- If a class performs more than one non-related functions, it is said to be lack of cohesion
- A lack of cohesion makes the overall structure of the software hard to manage, expand, maintain, and modify
- By improving information hiding you will generally be improving the coupling and cohesion
- Information hiding is the hiding of design decisions that are most likely to change (measured through Low Coupling and High Cohesion)

General Design Principles Principle of Low Coupling and High Cohesion

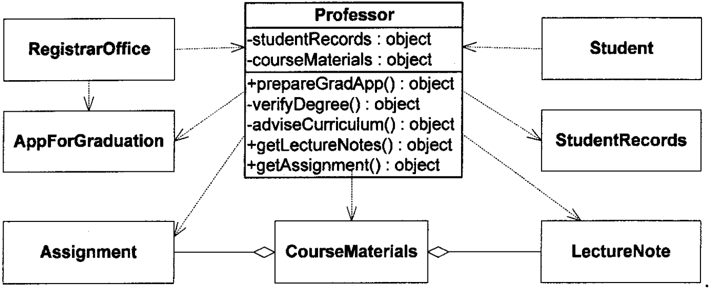
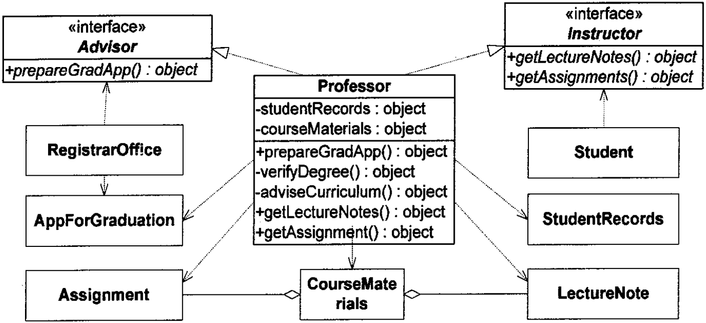


Figure: An initial design of a Professor class

General Design Principles Overview



«interface»
Advisor
+prepareGradApp() : object

Professor
-studentRecords : object
-courseMaterials : object
+prepareGradApp() : object
-verifyDegree() : object
-adviseCurriculum() : object
+getLectureNotes() : object
+getAssignment() : object

«interface»
Instructor
+getLectureNotes() : object
+getAssignments() : object

RegistrarOffice

AppForGraduation

Assignment

CourseMaterials

Student

StudentRecords

LectureNote

Figure: An improved design of a Professor class

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)

- The number of top-level packages in an architecture should be small

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)

- The number of top-level packages in an architecture should be small

- A range of 7 ± 2 is a useful guideline (projects might vary)

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)
- The number of top-level packages in an architecture should be small
- A range of 7 ± 2 is a useful guideline (projects might vary)
- The difference between small and large scale projects is the amount of nesting of modules or packages

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)
- The number of top-level packages in an architecture should be small
- A range of 7 ± 2 is a useful guideline (projects might vary)
- The difference between small and large scale projects is the amount of nesting of modules or packages
- Large scale projects typically organize each top-level package into subpackages

General Design Principles Principle of Low Coupling and High Cohesion

- Low coupled-lhigh cohesion architectures are far easier to modify (changes are more local)
- The number of top-level packages in an architecture should be small
- A range of 7 ± 2 is a useful guideline (projects might vary)
- The difference between small and large scale projects is the amount of nesting of modules or packages
- Large scale projects typically organize each top-level package into subpackages
- The 7 ± 2 guideline applies to each of these

General Design Principles Principle of Low Coupling and High Cohesion

One possible architecture for the most common video games consists of four packages.

- The environment in which the game takes place (areas, connections, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

One possible architecture for the most common video games consists of four packages.

- The environment in which the game takes place (areas, connections, etc.)
- The mechanism controlling the game (encounters, reactions to events, etc.)

Principle of Low Coupling and High Cohesion

One possible architecture for the most common video games consists of four packages.

- The environment in which the game takes place (areas, connections, etc.)
- The mechanism controlling the game (encounters, reactions to events, etc.)
- The participants in the game (player and foreign characters, etc.)

Principle of Low Coupling and High Cohesion

One possible architecture for the most common video games consists of four packages.

- The environment in which the game takes place (areas, connections, etc.)
- The mechanism controlling the game (encounters, reactions to events, etc.)
- The participants in the game (player and foreign characters, etc.)
- The artifacts involved in the game (swords, books, shields, etc.)

Principle of Low Coupling and High Cohesion

One possible architecture for the most common video games consists of four packages.

- The environment in which the game takes place (areas, connections, etc.)
- The mechanism controlling the game (encounters, reactions to events, etc.)
- The participants in the game (player and foreign characters, etc.)
- The artifacts involved in the game (swords, books, shields, etc.)

Each of these modules is quite cohesive

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)
- Loans (car, education, house, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)
- Loans (car, education, house, etc.)
- Investments (stocks, bonds, commodities, etc.)

Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)
- Loans (car, education, house, etc.)
- Investments (stocks, bonds, commodities, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)
- Loans (car, education, house, etc.)
- Investments (stocks, bonds, commodities, etc.)

Weaknesses: Little cohesion in the Accounts module

General Design Principles Principle of Low Coupling and High Cohesion

Consider how to decompose the design of a personal finance application

- Accounts (checking, savings, etc.)
- Bill paying (electronic, by check, etc.)
- Reports (total assets, liabilities, etc.)
- Loans (car, education, house, etc.)
- Investments (stocks, bonds, commodities, etc.)

Weaknesses: Little cohesion in the Accounts module

Great deal of coupling among these 5 parts

General Design Principles Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)

- Sources (employers, rental income, etc.)

Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)
- Sources (employers, rental income, etc.)
- Suppliers (landlord, loans, utilities, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)
- Sources (employers, rental income, etc.)
- Suppliers (landlord, loans, utilities, etc.)
- Interfaces (user interface, communications interface, reporting, etc.)

Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)
- Sources (employers, rental income, etc.)
- Suppliers (landlord, loans, utilities, etc.)
- Interfaces (user interface, communications interface, reporting, etc.)

General Design Principles Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)
- Sources (employers, rental income, etc.)
- Suppliers (landlord, loans, utilities, etc.)
- Interfaces (user interface, communications interface, reporting, etc.)

To understand which architecture options are better: experimental and investigative activity (try alternatives, modify them, and retry)

General Design Principles Principle of Low Coupling and High Cohesion

An alternative architecture

- Assets (checking accounts, stocks, bonds, etc.)
- Sources (employers, rental income, etc.)
- Suppliers (landlord, loans, utilities, etc.)
- Interfaces (user interface, communications interface, reporting, etc.)

To understand which architecture options are better: experimental and investigative activity (try alternatives, modify them, and retry)

Should be done at a high level (expensive at low level)

General Design Principles Open-Closed Principle

The principle urges 00 designers to meet two criteria:

- Open to extension: the system can be extended to meet new requirements.

General Design Principles Open-Closed Principle

The principle urges 00 designers to meet two criteria:

- Open to extension: the system can be extended to meet new requirements.
- Closed to modification: the existing implementation and code should not be modified as a result of system expansion

General Design Principles Open-Closed Principle

The principle urges 00 designers to meet two criteria:

- Open to extension: the system can be extended to meet new requirements.
- Closed to modification: the existing implementation and code should not be modified as a result of system expansion
- We should try our best to minimise the violation of this principle so that the reusability of the software can be maximised

General Design Principles Open-Closed Principle

The principle urges OO designers to meet two criteria:

- Open to extension: the system can be extended to meet new requirements.
- Closed to modification: the existing implementation and code should not be modified as a result of system expansion
- We should try our best to minimise the violation of this principle so that the reusability of the software can be maximised
- Technical approach for achieving Open-Closed Principle is the abstraction via inheritance and polymorphism

General Design Principles Open-Closed Principle

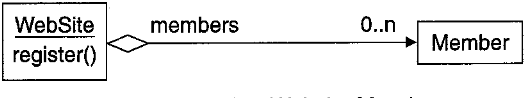


Figure: Registering Website Members (Rigid)

General Design Principles Open-Closed Principle

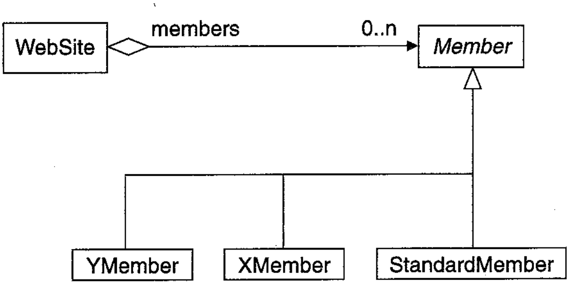


Figure: Registering Website Members (Flexible)

General Design Principles Open-Closed Principle

The Open-Closed Principle has many interesting implications

- Separation of interface and implementation

General Design Principles Open-Closed Principle

The Open-Closed Principle has many interesting implications

- Separation of interface and implementation
- Keep attributes private

General Design Principles Open-Closed Principle

The Open-Closed Principle has many interesting implications

- Separation of interface and implementation
- Keep attributes private
- Minimize the use of global variables

General Design Principles Open-Closed Principle

The Open-Closed Principle has many interesting implications

- Separation of interface and implementation
- Keep attributes private
- Minimize the use of global variables
- There are many other important OO design principles

Liskov substitution principle

Principle (Liskov substitution principle)

Let $q(x)$ be a property provable about objects x of type T. Then $q(y)$ should be true for objects y of type S where S is a subtype of T.

General Design Principles Liskov substitution principle

Principle (Liskov substitution principle)

Let $q(x)$ *be a property provable about objects* x *of type* T*. Then* $q(y)$ *should be true for objects* y *of type* S *where* S *is a subtype of* T*.*

General Design Principles Dependency Inversion Principle

Principle (Dependency Inversion Principle (DIP) /Inversion of Control)

High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions.

General Design Principles Dependency Inversion Principle

Principle (Dependency Inversion Principle (DIP) /Inversion of Control)

High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions.

This defines a very powerful rule for designing and programming: Design to an interface, not an implementation

General Design Principles Dependency Inversion Principle

Principle (Dependency Inversion Principle (DIP) /Inversion of Control (2))

Packages that are maximally stable should be maximally abstract. Instable packages should be concrete. The abstraction of a package should be in proportion to its stability.

General Design Principles Dependency Inversion Principle

Principle (Dependency Inversion Principle (DIP) /Inversion of Control (2))

Packages that are maximally stable should be maximally abstract. Instable packages should be concrete. The abstraction of a package should be in proportion to its stability.

In a sense, it follows what has been referred to as the Hollywood Principle: don't call us, we will call you

General Design Principles Interface Segregation Principle

Principle (Interface Segregation Principle)

Clients should not be forced to depend upon interfaces that they do not use.

General Design Principles Interface Segregation Principle

Principle (Interface Segregation Principle)

Clients should not be forced to depend upon interfaces that they do not use.

- It says: if there are two non-cohesive functionalities, keep them separate

General Design Principles Interface Segregation Principle

Principle (Interface Segregation Principle)

Clients should not be forced to depend upon interfaces that they do not use.

- It says: if there are two non-cohesive functionalities, keep them separate

- This avoids design of fat interfaces, and provides a clear design to the user (client)

General Design Principles Interface Segregation Principle

Principle (Interface Segregation Principle)

Clients should not be forced to depend upon interfaces that they do not use.

- It says: if there are two non-cohesive functionalities, keep them separate
- This avoids design of fat interfaces, and provides a clear design to the user (client)
- Break the functionalities into atomic interfaces that can be then individually accessed by the user

Law of Demeter

Principle (Law of Demeter)

Each unit should have only limited knowledge about other units: only units ”closely” related to the current unit.

General Design Principles Law of Demeter

Principle (Law of Demeter)

Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.

- It is a style rule for building systems

General Design Principles Law of Demeter

Principle (Law of Demeter)

Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.

- It is a style rule for building systems
- "Only talk to your immediate friends" is the motto

Law of Demeter

Principle (Law of Demeter)

Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.

- It is a style rule for building systems
- "Only talk to your immediate friends" is the motto
- Break the functionalities into atomic interfaces that can be then individually accessed by the user

Law of Demeter

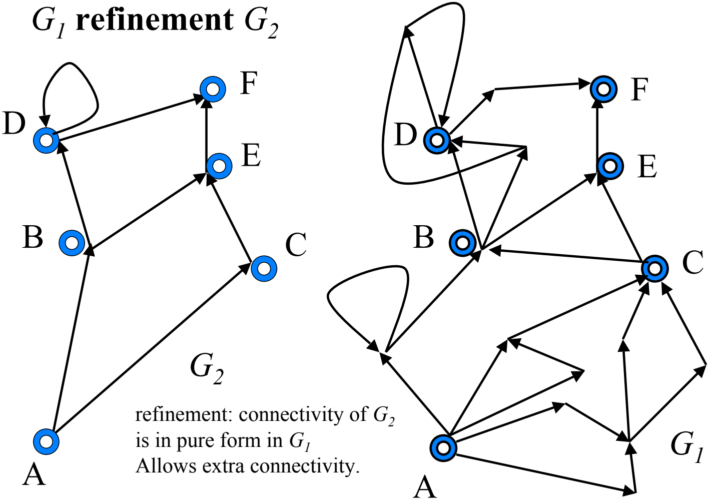
Principle (Law of Demeter)

Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.

- It is a style rule for building systems
- "Only talk to your immediate friends" is the motto
- Break the functionalities into atomic interfaces that can be then individually accessed by the user
- A method should have limited knowledge of an object model

General Design Principles Law of Demeter

G_1 refinement G_2



refinement: connectivity of G_2 is in pure form in G_1
Allows extra connectivity.

Other Design Principles for Security

Principle (Least Privilege)

The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task.

Other Design Principles for Security

Principle (Least Privilege)

The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task.

- If a subject does not need an access right, the subject should not have that right

Principle (Least Privilege)

The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task.

- If a subject does not need an access right, the subject should not have that right
- This is analogue to the "need to know" rule

Other Design Principles for Security

Principle (Fail-Safe Defaults)

The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.

General Design Principles Other Design Principles for Security

Principle (Fail-Safe Defaults)

The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.

- This is security version of this principle

General Design Principles Other Design Principles for Security

Principle (Fail-Safe Defaults)

The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.

- This is security version of this principle
- This principle assumes that the default access to an object is none

Principle (Fail-Safe Defaults)

The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.

- This is security version of this principle
- This principle assumes that the default access to an object is none

- If the subject is unable to complete its action or task, it should undo those changes it made in the security state of the system before it terminates

Principle (Fail-Safe Defaults)

The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.

- This is security version of this principle
- This principle assumes that the default access to an object is none

- If the subject is unable to complete its action or task, it should undo those changes it made in the security state of the system before it terminates

- Even if the program fails, the system is still safe

Principle (Economy of Mechanism)

The principle of economy of mechanism states that security mechanisms should be as simple as possible.

Other Design Principles for Security

Principle (Economy of Mechanism)

The principle of economy of mechanism states that security mechanisms should be as simple as possible.

- If a design and implementation are simple, fewer possibilities exist for errors

Principle (Economy of Mechanism)

The principle of economy of mechanism states that security mechanisms should be as simple as possible.

- If a design and implementation are simple, fewer possibilities exist for errors

- This principle simplifies the design and implementation of security mechanisms

Other Design Principles for Security

Principle (Economy of Mechanism)

The principle of economy of mechanism states that security mechanisms should be as simple as possible.

- If a design and implementation are simple, fewer possibilities exist for errors
- This principle simplifies the design and implementation of security mechanisms
- Simple design $\Longrightarrow$ less assumptions $\Longrightarrow$ less risks

Other Design Principles for Security

Principle (Economy of Mechanism)

The principle of economy of mechanism states that security mechanisms should be as simple as possible.

- If a design and implementation are simple, fewer possibilities exist for errors
- This principle simplifies the design and implementation of security mechanisms
- Simple design $\Longrightarrow$ less assumptions $\Longrightarrow$ less risks
- Simple design $\Longrightarrow$ simpler testing

Other Design Principles for Security

Principle (Complete Mediation)

The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.

General Design Principles Other Design Principles for Security

Principle (Complete Mediation)

The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.

- This principle restricts the caching of information

General Design Principles Other Design Principles for Security

Principle (Complete Mediation)

The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.

- This principle restricts the caching of information
- When a subject attempts to read an object, the operating system should mediate the action (determines if he is allowed + provides the resources)

General Design Principles Other Design Principles for Security

Principle (Complete Mediation)

The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.

- This principle restricts the caching of information
- When a subject attempts to read an object, the operating system should mediate the action (determines if he is allowed + provides the resources)
- If the subject tries to read the object again, the system should check that the subject is still allowed to read the object

General Design Principles Other Design Principles for Security

Principle (Open Design)

The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.

Principle (Open Design)

The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.

- This principle suggests that complexity does not add security

Principle (Open Design)

The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.

- This principle suggests that complexity does not add security
- If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism ("security through obscurity")

Principle (Open Design)

The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.

- This principle suggests that complexity does not add security
- If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism ("security through obscurity")
- This is especially true of cryptographic software and systems (algorithms kept secret)

Other Design Principles for Security

Principle (Open Design)

The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.

- This principle suggests that complexity does not add security
- If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism ("security through obscurity")
- This is especially true of cryptographic software and systems (algorithms kept secret)
- Keeping cryptographic keys and passwords secret does not violate this principle

General Design Principles Other Design Principles for Security

Principle (Separation of Privilege)

The principle of separation of privilege states that a system should not grant permission based on a single condition.

Principle (Separation of Privilege)

The principle of separation of privilege states that a system should not grant permission based on a single condition.

- This principle is restrictive because it limits access to system entities

Principle (Separation of Privilege)

The principle of separation of privilege states that a system should not grant permission based on a single condition.

- This principle is restrictive because it limits access to system entities
- This principle is equivalent to the separation of duty principle

Principle (Separation of Privilege)

The principle of separation of privilege states that a system should not grant permission based on a single condition.

- This principle is restrictive because it limits access to system entities
- This principle is equivalent to the separation of duty principle
- Systems and programs granting access to resources should do so only when more than one condition is met

Other Design Principles for Security

Principle (Least Common Mechanism)

The principle of least common mechanism states that mechanisms used to access resources should not be shared.

Other Design Principles for Security

Principle (Least Common Mechanism)

The principle of least common mechanism states that mechanisms used to access resources should not be shared.

- Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized

Principle (Least Common Mechanism)

The principle of least common mechanism states that mechanisms used to access resources should not be shared.

- Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized
- This principle is restrictive because it limits sharing

Principle (Psychological Acceptability)

The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

Principle (Psychological Acceptability)

The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

- It recognizes the human element in security

General Design Principles Other Design Principles for Security

Principle (Psychological Acceptability)

The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

- It recognizes the human element in security
- Configuring and executing a program should be as easy and as intuitive as possible

Other Design Principles for Security

Principle (Psychological Acceptability)

The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

- It recognizes the human element in security
- Configuring and executing a program should be as easy and as intuitive as possible
- In practice, the principle of psychological acceptability is interpreted to mean that the security mechanism may add some extra burden, but that burden must be both minimal and reasonable