Χαρακτηριστικά ενοτήτων

Στόχοι

- Πώς να δημιουργήσεις ενότητες υψηλής ποιότητας στην C.
- Γιατί χρειάζεται;
  - Η αφαίρεση είναι απαραίτητη τεχνική για την ανάπτυξη και κατανόηση μεγάλων, πολύπλοκλων συστημάτων
  - Ο καλός προγραμματιστής γνωρίζει πώς να βρει κατάλληλες αφαιρέσεις σε ένα μεγάλο πρόγραμμα
  - Ο καλός προγραμματιστής γνωρίζει πώς να παρουσιάζει τις αφαιρέσεις σε ένα μεγάλο πρόγραμμα μέσω των ενοτήτων.

Χαρακτηριστικά Ενοτήτων

- Μια καλά σχεδιασμένηενότητα
  1. Διαχωρίζει διεπαφή και υλοποίηση (μερική απόκρυψη)
  2. Ενθυλακώνει δεδομένα (ολική απόκρυψη)
  3. Διαχειρίζεται πόρους με συνέπεια
  4. Κάνει καλή επιλογή Ονομάτων
  5. Έχει ελάχιστη διεπαφή
  6. Αναφέρει λάθη στους πελάτες
  7. Ορίζει συμβόλαια
  8. Έχει υψηλή συνάφεια (μεταξύ των πράξεων)
  9. Έχει ασθενή εξάρτηση (από άλλες ενότητες)

1. Διαχωρίζει διεπαφή και υλοποίηση

- Αποκρύπτει την υλοποίηση συναρτήσεων από τους πελάτες (υλοποίηση αφαίρεσης)
- Επιτρέπει την ανεξάρτητη μεταγλώττιση

Παράδειγμα

Stack: Στοίβα που διαχειρίζεται strings

- Σχεδιαστική Επιλογή με Δυναμική Συνδεδεμένη Λίστα.
- Πράξεις
  - new: Δημιουργία νέου Stack
  - free: Απελευθέρωση
  - push: ώθηση
  - top: εξαγωγή στοιχείου κορυφής
  - pop: απόρριψη στοιχείου κορυφής
  - isEmpty: Return 1 αν η Stack κενό

Χωρίς απόκρυψη

```
/* stack.c */
struct Node {
   const char *item;
   struct Node *next;
};
struct Stack {
   struct Node *first;
};
struct Stack *Stack_new(void) {…}
void  Stack_free(struct Stack *s) {…}
void  Stack_push(struct Stack *s, const char *item) {…}
char *Stack_top(struct Stack *s) {…}
void  Stack_pop(struct Stack *s) {…}
int   Stack_isEmpty(struct Stack *s) {…}
```

```
/* client.c */

#include "stack.c"

/* Use the functions
defined in stack.c. */
```

- Δεν υπάρχει διεπαφή (υπάρχει μόνο ένα αρχείο)
- Αλλαγή stack.c => rebuild stack.c και πελάτη
- Χωρίς αφαίρεση, ορισμοί με αχρείαστες λεπτομέρειες είναι ορατοί.

Η Μερική Απόκρυψη (διεπαφή)

Ενότητα Stack αποτελείται από δυο αρχεία

- Το stack.h (the interface) δηλώνει functions and ορίζει data structures
  (ο τύπος στοιχείου είναι char*)

```
/* stack.h */

struct Node {
   const char *item;
   struct Node *next;
};
struct Stack {
   struct Node *first;
};
struct Stack *Stack_new(void);
void  Stack_free(struct Stack *s);
void  Stack_push(struct Stack *s, const char *item);
char *Stack_top(struct Stack *s);
void  Stack_pop(struct Stack *s);
int   Stack_isEmpty(struct Stack *s);
```

Μερική Απόκρυψη (υλοποίηση)

Το stack.c (η υλοποίηση) ορίζει functions

- #include “stack.h”
  - Έλεγχος μεταγλωττιστή
  - Οι συναρτήσεις έχουν πρόσβαση στα δεδομένα

```
/* stack.c */
#include "stack.h"

struct Stack *Stack_new(void) {…}
void  Stack_free(struct Stack *s) {…}
void  Stack_push(struct Stack *s, const char *item) {…}
char *Stack_top(struct Stack *s) {…}
void  Stack_pop(struct Stack *s) {…}
int   Stack_isEmpty(struct Stack *s) {…}
```

Μερική Απόκρυψη (χρήση)

- Πελάτης #include την διεπαφή
- Αν αλλάξει το stack.c => μεταγλώττιση μόνο του stack.c, όχι του πελάτη.
- Καλύτερη αφαίρεση (οι προσβάσεις στην δομή μπορούν να γίνουν μέσω συναρτήσεων)

```
/* client.c */

#include "stack.h"

/* Use the functions declared in stack.h. */
```

2. Ενθυλάκωση Δεδομένων (Ολική Απόκρυψη)

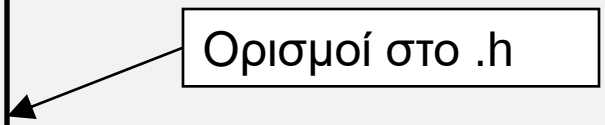
Η πρόσβαση στη δομή αποκλειστικά μέσω συναρτήσεων
  - Πληρης απόκρυψη λεπτομερειών υλοποίησης
  - Συναρτήσεις για πρόσβαση
  - Οι πελάτες δεν έχουν πρόσβαση απευθείας στα δεδομένα.
- Όφελος:
  - Ευκρίνεια - μέσω της αφαίρεσης
  - Ασφάλεια - οι πελάτες δεν μπορούν να καταστρέψουν αντικείμενα
  - Ευελιξία – Επιτρέπει αλλαγές στην υλοποίηση, ακόμα και στα δεδομένα, χωρίς να επηρεάζονται οι πελάτες.

Πρόβλημα με Μερική απόκρυψη

```
/* stack.h */

struct Node {
   const char *item;
   struct Node *next;
};
struct Stack {
   struct Node *first;
};

struct Stack *Stack_new(void);
void  Stack_free(struct Stack *s);
void  Stack_push(struct Stack *s, const char *item);
char *Stack_top(struct Stack *s);
void  Stack_pop(struct Stack *s);
int   Stack_isEmpty(struct Stack *s);
```



- Διεπαφή αποκαλύπτει υλοποίηση με συνδεδεμένη λίστα
- Ο πελάτης μπορεί να αλλάξει/προσπελάσει απευθείας τα δεδομένα, πιθανά να τα αλλοιώσει.
- struct Stack S; … S->item = NULL;

Ολική Απόκρυψη

```
/* stack.h */

typedef struct Stack * Stack_T;

Stack_T Stack_new(void);
void  Stack_free(Stack_T s);
void  Stack_push(Stack_T s, const char *item);
char *Stack_top(Stack_T s);
void  Stack_pop(Stack_T s);
int   Stack_isEmpty(Stack_T s);
```

- Μετακίνηση ορισμού στο .c και χρήση δείκτη
  - Αδιαφανής (opaque) δείκτης Stack_T
  - “Stack_T” ο νέος τύπος
  - Ο πελάτης δεν έχει πρόσβαση στα δεδομένα άμεσα; Τα δεδομένα είναι αδιαφανή.
  - Stack_T S; … S->item=NULL (syntax error).

Stdio

```
/* stdio.h */

struct FILE {
   int cnt;     /* characters left */
   char *ptr;   /* next character position */
   char *base;  /* location of buffer */
   int flag;    /* mode of file access */
   int fd;      /* file descriptor */
};
…
```

- Μερική απόκρυψη
- Αλλά δεν χρειάζεται να δούμε τους ορισμούς

3. Διαχειρίζεται πόρους με συνέπεια

- Ελευθερώνει πόρους μόνο αν τους έχει δεσμεύσει. Π.χ
  - malloc <=> free
  - opens file <=> closes file
- Δέσμευση και αποδέσμευση σε διαφορετικά επίπεδα έχει κινδύνους
  - Δεν κάνουμε free => memory leak (διαρροή)
  - Δεν κάνουμε malloc => dangling pointer (αιωρούμενος), seg fault
  - Δεν κλείνουμε close file => μη αποδοτική χρήση πόρου
  - Δεν ανοίγουμε το open file => dangling pointer (αιωρούμενος), seg fault

Παράδειγμα

- Stack: Ποιος δεσμεύει και αποδεσμεύει τα strings;
  - Αποδεκτές Επιλογές

    (1) Client allocates and frees strings
    - **`Stack_push()`** δεν δημιουργεί string, δέχεται δείκτη.
    - **`Stack_pop()`** δεν ελευθερώνει το string
    - **`Stack_free()`** δεν ελευθερώνει τα strings

    (2) Stack object allocates and frees strings
    - **`Stack_push()`** δημιουργεί string
    - **`Stack_pop()`** ελευθερώνει το string
    - **`Stack_free()`** ελευθερώνει τα strings
  - Η επιλογή μας η (1) για να έχουμε γενικό ορισμό του Stack.
  - Μη αποδεκτές επιλογές:
    - Ο πελάτης δημιουργεί string , Stack ελευθερώνει
    - Stack δημιουργεί strings, ο πελάτης ελευθερώνει

4. Καλή επιλογή Ονομάτων

Ονόματα με συνέπεια

- Όνομα συνάρτησης περιέχει το όνομα της δομής
  - Βοηθάει στην συντήρηση του προγράμματος
  - Μειώνει πιθανότητα σύγκρουσης ονομάτων
- Οι συναρτήσεις να έχουν συνεπή διάταξη παραμέτρων
  - Βοηθάει στην κλήση τους

Παραδείγματα

- Stack
  - (+) Συναρτήσεις ξεκινούν με πρόθεμα “Stack_”
  - (+) πρώτη παράμετρος τύπος δομής
- string
  - (+) ξεκινούν με “str”
  - (+) Προορισμός πρώτη, προέλευση δεύτερη; Μιμείται απόδοση τιμής
- stdio
  - (-) Μερικές συναρτήσεις ξεκινούν “f”, άλλες όχι
  - (-) Μερικές συναρτήσεις έχουν πρώτη παράμετρο FILE. Άλλες (e.g. **`putc()`**) διαφορετική.

5. Έχει ελάχιστη διεπαφή

- Δήλωση στο .h εφόσον
  - Είναι απαραίτητη ή
  - Βολική
- Περισσότερες συναρτήσεις σημαίνει υψηλότερο κόστος συντήρησης και χρόνο μάθησης

Παράδειγμα Στοίβα

```
/* stack.h */

typedef struct Stack *Stack_T ;

Stack_T Stack_new(void);
void  Stack_free(Stack_T s);
void  Stack_push(Stack_T s, const char *item);
char *Stack_top(Stack_T s);
void  Stack_pop(Stack_T s);
int   Stack_isEmpty(Stack_T s);
```

– Όλες απαραίτητες

Παράδειγμα Stack

- Αν προσθέταμε την

```
void Stack_clear(Stack_T s);
```

  - Ακυρώνει όλα τα στοιχεία
  - Δεν είναι απαραίτητη; Ο πελάτης μπορεί με συνεχόμενα **`pop()`** να επιτύχει το ίδιο
  - Ίσως βολική

Παράδειγμα String

```
/* string.h */
/* απαραίτητες*/
size_t strlen(const char *s);
char  *strncpy(char *dest, const char *src, size_t n);
char  *strncat(char *dest, const char *src, size_t n);
char  *strncmp(const char *s, const char *t, size_t n);
char  *strstr(const char *haystack, const char *needle);

/* βολικές και αποδοτικές */
char  *strcpy(char *dest, const char *src);
char  *strcat(char *dest, const char *src);
char  *strcmp(const char *s, const char *t);
```

Παράδειγμα stdio

```
/* απαραίτητες */
FILE *fopen(const char *filename, const char *mode);
int   fclose(FILE *f);
int   fflush(FILE *f);
int   fgetc(FILE *f);
int   putc(int c, FILE *f);
int   fscanf(FILE *f, const char *format, …);
int   fprintf(FILE *f, const char *format, …);

/* βολικές */
int   getchar(void);
int   printf(const char *format, …);
int   putchar(int c);
int   scanf(const char *format, …);

/* τίποτα από τα δυο */
int   getc(FILE *f);
…
```

6. Αναφέρει λάθη

Αναφέρει λάθη στους πελάτες

– Η ενότητα διαπιστώνει λάθη, το οποία χειρίζεται ο πελάτης

Τα λάθη τα χειρίζεται καλύτερα ο πελάτης

Αναφορά λαθών στην C

- Εντοπισμός λάθους
  - **`if`** statement
  - **`assert`** macro
- Αναφορά στον πελάτη
  - Με καθολική μεταβλητή
    - Ξεχνάμε να ελέγξουμε (δεν φαίνεται άμεσα)
    - Δεν ενδείκνυται για πολυνηματικές εφαρμογές
  - Επιστροφή τιμής
    - Ποια τιμή να επιλέξουμε για λάθος
  - Επιπλέον παράμετρος-σημαία (flag)
    - Μια επιπλέον παράμετρος
  - Κλήση **`assert`** macro
    - Εντοπίζει, αλλά δεν ανακάμπτει
- Δεν υπάρχει ιδανική επιλογή

Αναφορά λαθών στην C (συνεχ.)

- Διαφοροποιούμε
- Λάθη χρήστη (User errors)
  - Errors made by human user
    - Example: Bad data in stdin
    - Example: Bad command-line argument
  - Errors that “easily could happen”
  - To detect: Use **`if`** statement
  - To report: Use return value or (by-pointer-value) parameter
- Λάθη Προγραμματιστή (Programmer errors)
  - Errors made by a programmer
  - Errors that “never should happen”
    - Example: Call **`Stack_pop()`** with NULL stack, empty stack
  - To detect and report: Use **`assert`**
- The distinction sometimes is unclear
  - Example: Write to file fails because disk is full

Αναφορά λαθών στην C (συνεχ.)

- Stack

```
/* stack.c */

…

void Stack_push(Stack_T s, const char *item) {
   struct Node *p;
   assert(s != NULL);
   p = (struct Node*)malloc(sizeof(struct Node));
   assert(p != NULL);
   p->item = item;
   p->next = s->first;
   s->first = p;
}
```

  - Stack functions:
    - Consider invalid parameter to be **programmer** error
    - Consider **`malloc()`** failure to be **programmer** error
    - Detect/report no **user** errors

Παραδείγματα

- string
  - No error detection or reporting
    - Example: NULL parameter to **`strlen()`** => probable seg fault
- stdlib
  - Uses return values to indicate failure
    - Note awkwardness of **`scanf()`**
  - Sets global variable “errno” to indicate cause of failure

7. Ορίζει συμβόλαια

  - A module should establish contracts with its clients
  - Contracts should describe what each function does, esp:
    - Meanings of parameters
    - Valid/invalid parameter values
    - Meaning of return value
    - Side effects
- Οι λόγοι
  - Establishing contracts facilitates cooperation between multiple programmers on a team
  - Establishing contracts assigns blame to violators
    - Catch errors at the door!
    - Better that the boss yells at the programmer who is your client rather than at you!!!

Ορισμός συμβολαίων στην C

- Προτείνουμε
- Με σχόλια
  - Η υλοποίηση ακολουθεί τα σχόλια

Παράδειγμα

- Stack

```
/* stack.h */
…
char *Stack_top(Stack_T s);
/* Return the top item of stack s.
   It is a checked runtime error for s
   to be NULL or empty. */
…
```

  - Comment defines contract:
    - Meanings of function's parameters
      - s is the pertinent stack
    - Valid/invalid parameter values
      - s cannot be NULL or empty
    - Meaning of return value
      - The return value is the top item
    - Side effects
      - (None, by default)

8. Συνάφεια

Υψηλή Συνάφεια

- Οι συναρτήσεις να έχουν σχέση μεταξύ τους

- Βοηθάει την αφαίρεση

Υψηλή Συνάφεια – Παραδείγματα

- Stack

  (+)All functions are related to the encapsulated data

- string

  (+) Most functions are related to string handling

  (-) Some functions are not related to string handling

  `memcpy()`, **`memmove()`**, **`memcmp()`**, **`memchr()`**, **`memset()`**

  (+) But those functions are similar to string-handling functions

- stdio

  (+) Most functions are related to I/O

  (-) Some functions don't do I/O

  `sprintf()`, **`sscanf()`**

  (+)But those functions are similar to I/O functions

9. Ασθενής Εξάρτηση

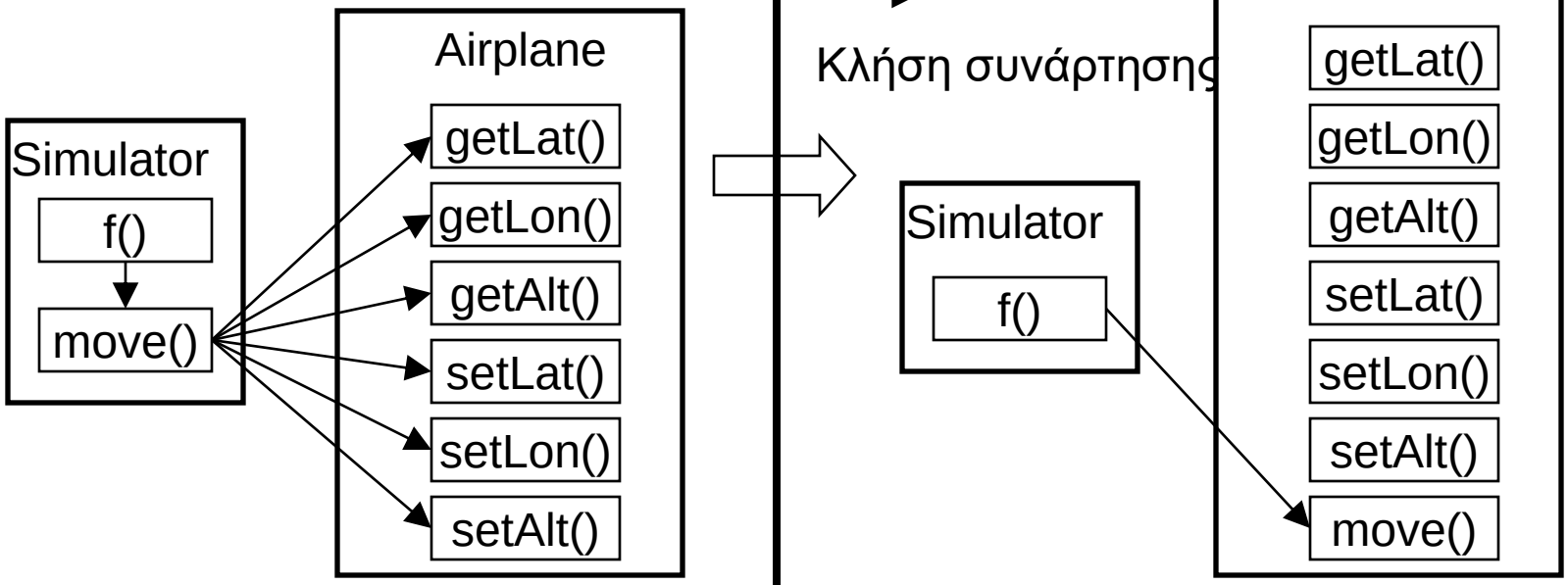
Τι σημαίνει

- Να συνδέεται λίγο με άλλες ενότητες
- Διάδραση εσωτερικά σε ενότητες πιο έντονες από διάδραση μεταξύ ενοτήτων

- Παρατηρήσεις
  - Συντήρηση : Με ασθενή εξάρτηση το πρόγραμμα αλλάζει πιο εύκολα
  - Επαναχρησιμοποίηση: Η ασθενή εξάρτηση βοηθάει στην επαναχρησιμοποίηση σε άλλα προγράμματα
- Εμπειρία
  - Ενότητες έχουν λιγότερα λάθη

Ασθενής Εξάρτηση Παράδειγμα

- Σχεδιασμός



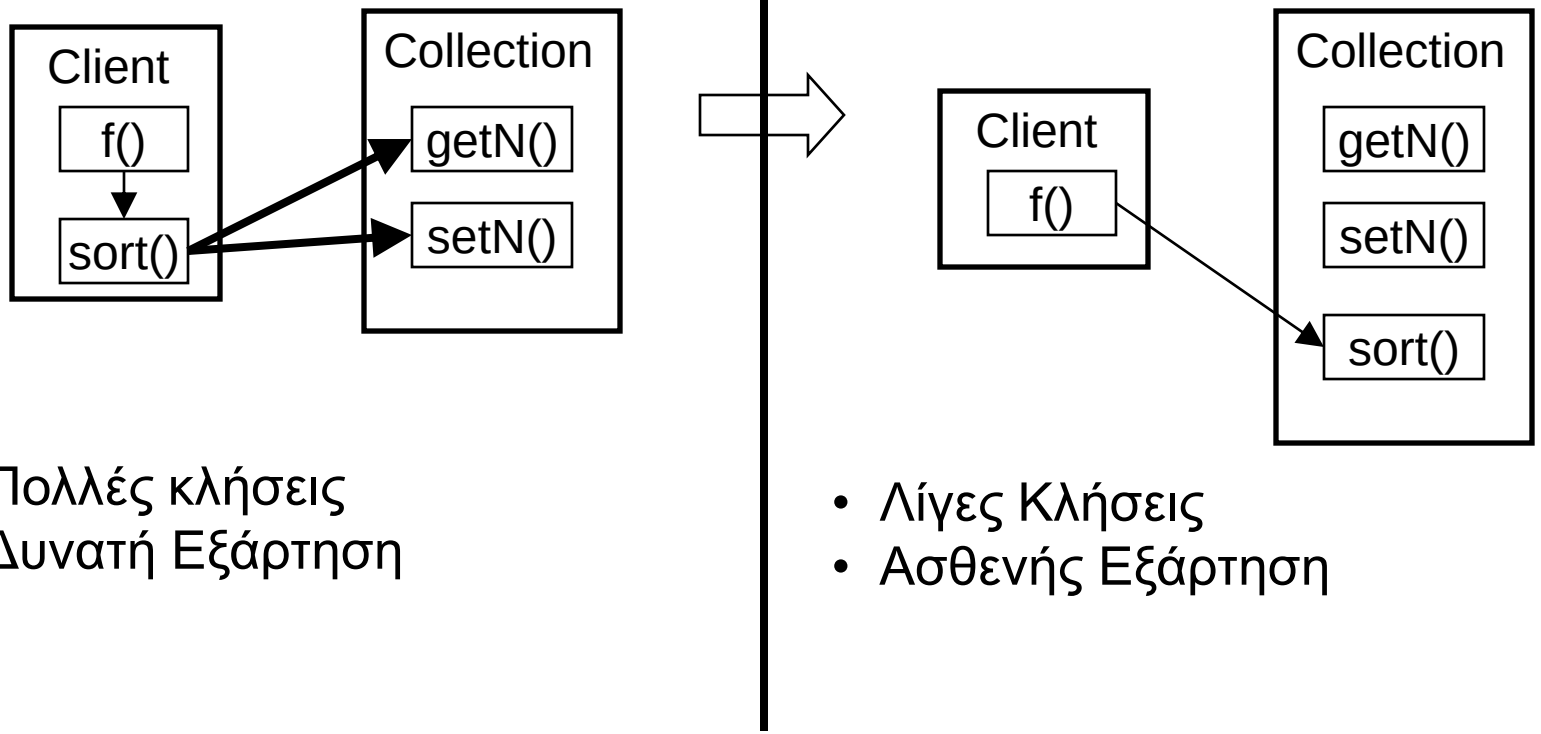
- Πελάτης καλεί πολλές συναρτήσεις
- Δυνατή Εξάρτηση - σχεδιασμός

- Πελάτης καλεί λίγες συναρτήσεις
- Ασθενής εξάρτηση - σχεδιασμός

Ασθενής Εξάρτηση-Παράδειγμα

→ Many function calls → One function call

- Κατά την εκτέλεση-ασθενής εξάρτηση

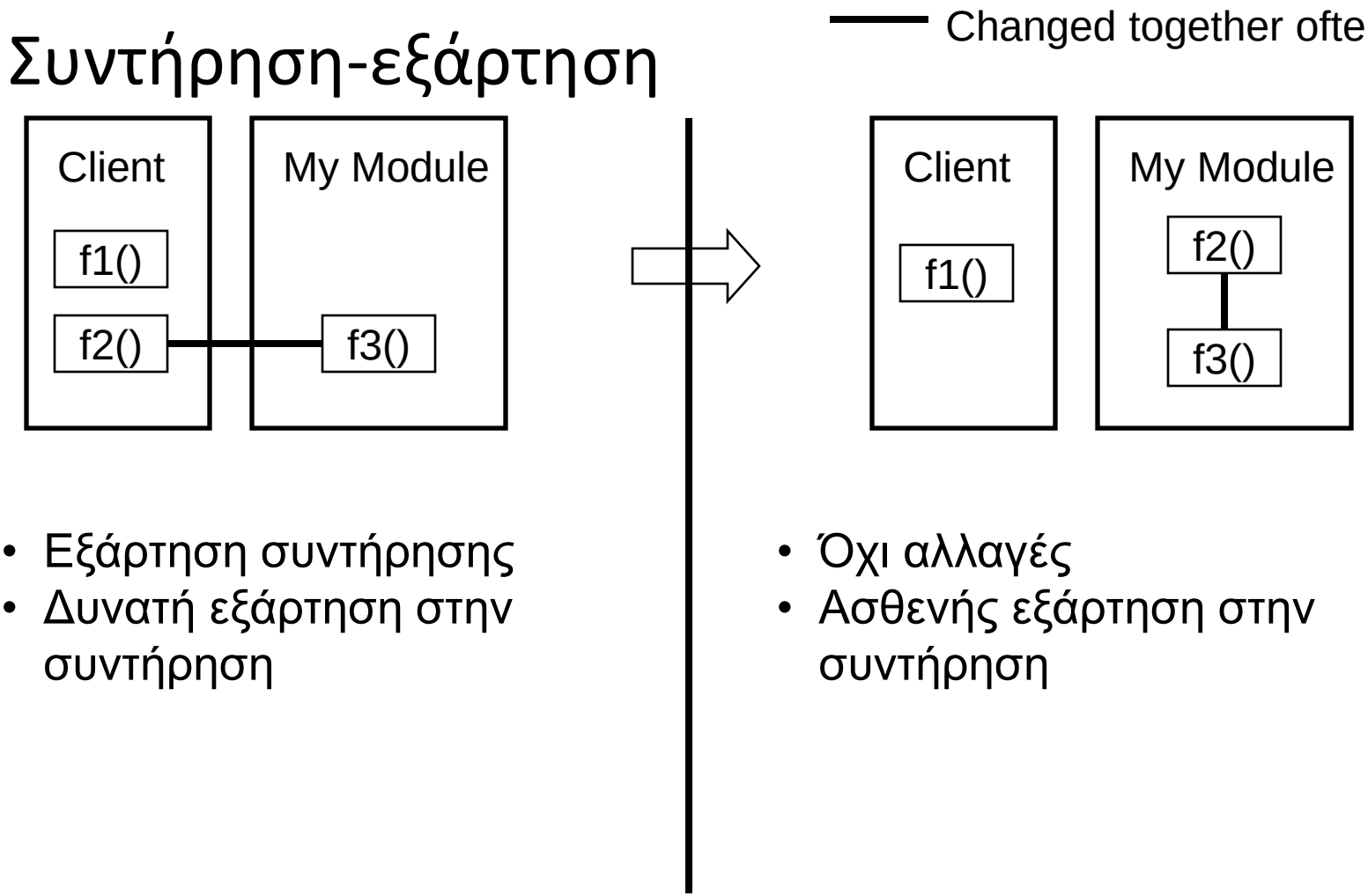


- Πολλές κλήσεις
- Δυνατή Εξάρτηση

- Λίγες Κλήσεις
- Ασθενής Εξάρτηση

Ασθενής Εξάρτηση-Παράδειγμα

- Συντήρηση-εξάρτηση



- Εξάρτηση συντήρησης
- Δυνατή εξάρτηση στην συντήρηση

- Όχι αλλαγές
- Ασθενής εξάρτηση στην συντήρηση

Επιτυγχάνοντας Ασθενή Εξάρτηση

Μετακίνηση κώδικα

- Από πελάτες σε ενότητα
- Από ενότητα σε πελάτες
- Από πελάτες και ενότητες σε νέες ενότητες