Estruturas de Dados / Programação 2
Tipos Abstratos de Dados (TADs)

Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

If you were the team leader…

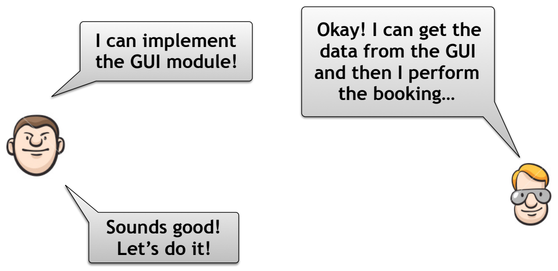
- How many programmers?
- How many modules?
- How many months to finish this software?
- Remember: the software must be deployed as soon as possible!

How to deal with several developers?

Parallel development

Scenario: two developers

- They are talking about how they will implement the hotel booking system…


I can implement the GUI module!
Okay! I can get the data from the GUI and then I perform the booking…
Sounds good! Let's do it!

Booking module

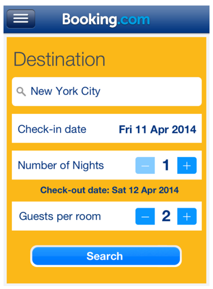
- Developer 1 starts writing his module…

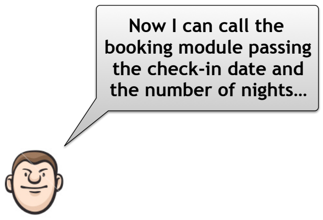
```
void book(int check_in_day,
          int check_in_month,
          int check_in_year,
          int check_out_day,
          int check_out_month,
          int check_out_year)
{
  …
  //single room
  …
  //double room
  …
}
```

Okay… I get the data from the GUI, then I ask for the room type…

GUI module

- Developer 2 starts writing his module…




Now I can call the booking module passing the check-in date and the number of nights…

Conflict

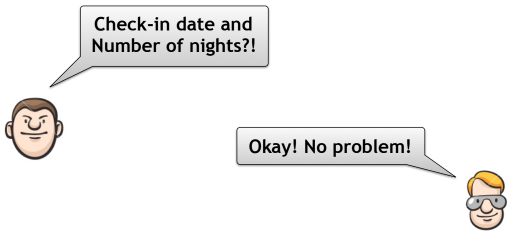
```
void book(int check_in_day,
          int check_in_month,
          int check_in_year,
          int check_out_day,
          int check_out_month,
          int check_out_year)
{
  …
}
```

```
book(11, 4, 2014, 1);
```

We need a contract!

To develop in parallel…

- … developers need to enforce **contracts** before implementing the modules



Using the contract, we do not have conflicts!

```
void book(int check_in_day,
          int check_in_month,
          int check_in_year,
          int number_of_nights)
{
  …
}
```

```
book(11, 4, 2014, 1);
```

Modules

Modules

- They are essential for big systems
- Only one module: big and complex task
  - Difficult to implement
  - Difficult to test
- We divide the task into small modules
  - Easier to implement
  - Easier to test

Example in C: header files

```
/*
  Returns the number of characters of str.
*/
int length(char *str);

/*
  Concatenates string “from” into “to”.
*/
void concat(char *to, char *from);
```

Comments (Documentation)

- Document the functions offered by the module
- How developers can use the module
- Remember our search algorithm?
  - Returns -1 when the element was not found…
  - Important information for those who call this algorithm!

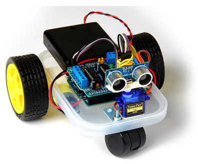
Using the module interface

```
#include <stdio.h>
#include “str.h”

int main()
{
  // using length and concat…
}
```

Another example

Obstacle avoiding robot



```
int get_distance()
{
  uS = ultrasonic.ping();
  return uS / US_ROUNDTRIP_CM;
}

void robot()
{
  distance_in_cm = get_distance();

  if (distance_in_cm <= SAFE_DISTANCE) {
    stop();
    look_around();
  }
  …
}
```

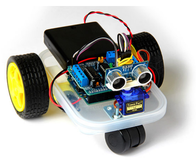
No obstacles, the robot stops… why?!

NewPing

```
//Value returned if there's no ping echo within the
//specified MAX_SENSOR_DISTANCE or max_cm_distance.
#define NO_ECHO 0

/*
 Trigger a ping, if it returns false,
 return NO_ECHO to the calling function.
*/
unsigned int NewPing::ping()
{
   if (!ping_trigger()) return NO_ECHO;
  …
}
```

Condition according to the NewPing documentation…



```
int get_distance()
{
  uS = ultrasonic.ping();
  return uS / US_ROUNDTRIP_CM;
}

void robot()
{
  distance_in_cm = get_distance();

  if ((distance_in_cm <= SAFE_DISTANCE) &&
      (distance_in_cm != 0)) {
    stop();
    look_around();
  }
  …
}
```

Abstract Data Types (ADTs)

Module

- Encompasses
  - Functions
  - Related functionalities
  - Well defined end

Abstract Data Type: module that defines…

New data type

+

Set of operations to manipulate data of this type

Example: Point ADT

- We want to define a new type: Point
- Developers should declare this new type and use it…
- Which operations can we do when using points?

Point ADT

```
point* create_point(float x, float y);

void free_point(point *point);

void get_point(point *point, float *x, float *y);

void set_point(point *point, float x, float y);

float points_distance(point *point1, point *point2);
```

Client using our new Point type…

```
#include <stdio.h>
#include "point.h"

int main()
{
  point *point1 = create_point(1.0, 1.0);
  point *point2 = create_point(5.0, 4.0);

  float distance = points_distance(point1, point2);

  printf("Distance = %f\n", distance);
}
```

Abstraction

- Usually we do not care about **how** the module was implemented!
- So, we hide the strategy used in the implementation

~~How~~ What

```
struct point {
  float x;
  float y;
};

point* create_point(float x, float y)
{
  point *new_point = (point*) malloc(sizeof(point));
  if (new_point == NULL) {
    printf("Insufficient Memory!");
    exit(1);
  }
  new_point->x = x;
  new_point->y = y;
  return new_point;
}
```

Now, we can change the implementation and…

… as long as we keep the contract…

```
float points_distance(point *point1, point *point2)
{
  //new fantastic and precise method to compute
  //the distance between two points…
}
```

… this new implementation **DOES NOT** affect the client code!

```
int main()
{
  float distance = points_distance(point1, point2);
  …
}
```

Advantages

- Reuse
- Improve maintenance tasks
- Improve developers productivity
- Better time-to-market
- Can you see any disadvantage?!

The Point ADT does not export the struct point…

- So, the client cannot access such a struct…

```
#include <stdio.h>
#include "point.h"

int main() { … }
```

```
struct point {
  float x;
  float y;
};
```

- Clients that use the Point ADT cannot access data ("x" and "y") directly

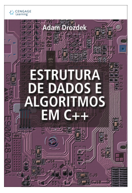
- However, we can do it by using the functions!
  - createPoint
  - getPoint / setPoint

Encapsulation to avoid undesirable data

```
struct circle {
  point *point;
  float radius;
};

circle* create_circle(point *point, float radius)
{
  if (radius <= 0) {
    printf("Radius must be greater than zero");
    exit(1);
  }
  …
  return new_circle;
}
```

References



Chapter 1