Software Improvement Group

Challenges of Secure Coding

Bárbara Vieira

BSides Lisbon 2016

November 11, 2016

What's our mission?

GETTING SOFTWARE RIGHT

We help clients improve their software with the aim of

Reducing costs • Improving speed • Ensuring software's continuity

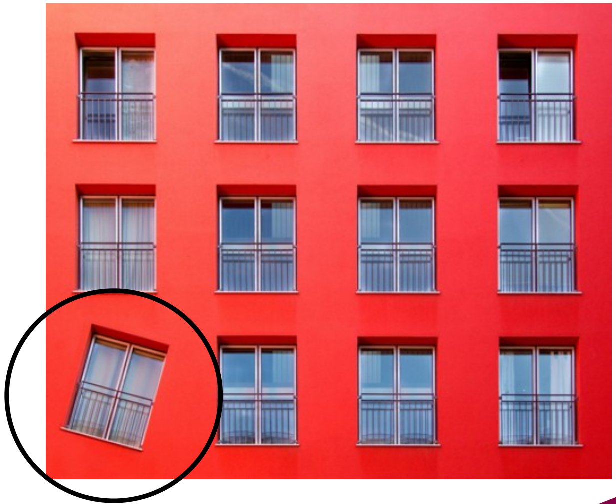
We are an international consultancy agency aimed at software improvement founded in 2000.

We are an independent, objective and impartial mediator between clients and IT suppliers.

We deliver management insight in IT project and systems based on validated facts and measurements.

We actively contribute to scientific research in the field of software engineering.

Security by design and by default



Security by design and by default failover

Fail to deploy on time or threats are exposed

Training

Requirements

Design

Coding

Verification and testing

Secure Software Development Lifecycle

Security Requirements

"Software needs to be secure"

Failing identifying and deriving countermeasures!

Failing verifying the requirements and lack of proper testing

Security as an overlay

What do we do?

```
<TABLE width="100%" cellpadding="3" cellspacing="0" Border="1" bgcolor="#ff

<TR align="center" valign="top">
<TD width="30%">This is an example of using a Table, without border,
                for arranging text and images spatially on the page.</TD>
<TD width="20%">  Note that in this example
                the space has been divided</TD>
<TD width="20%">into 4 columns. The image below  uses 2 of them.</TD>
<TD width="30%"> We also use cells that extend over several rows.
        </TD></TR>
<TR>
<TD rowspan="2"><font face="arial, verdana, geneva, helvetica"> Here we are
        to put a description of the city for which the map is shown on the
        <br>Note: we have specified a different font type for this cell.</T
  <TD colspan="2" rowspan="2" align="center">
 <IMG width="200" height="143" align="center" hspace="5" vspace="5"
          SRC="photoalbum/au-salzplanm.jpg">
 </TD>
 <TD>More text about the town</TD>
 </TR>
 <TR>
 <TD align="center" bgcolor="#ffcccc"><font size="+1">This row has only 1 c
        The other columns occupied the remaining cells.</TD></TR>
 <TR><TD colspan="4">
        Here's a cell which extends all across the page,
        all 4 columns, for example a page footer. Finally
        we must put the "End Table" statement   &#60;/TABLE&#62;</TD>
  </TR>
  </TABLE>
```

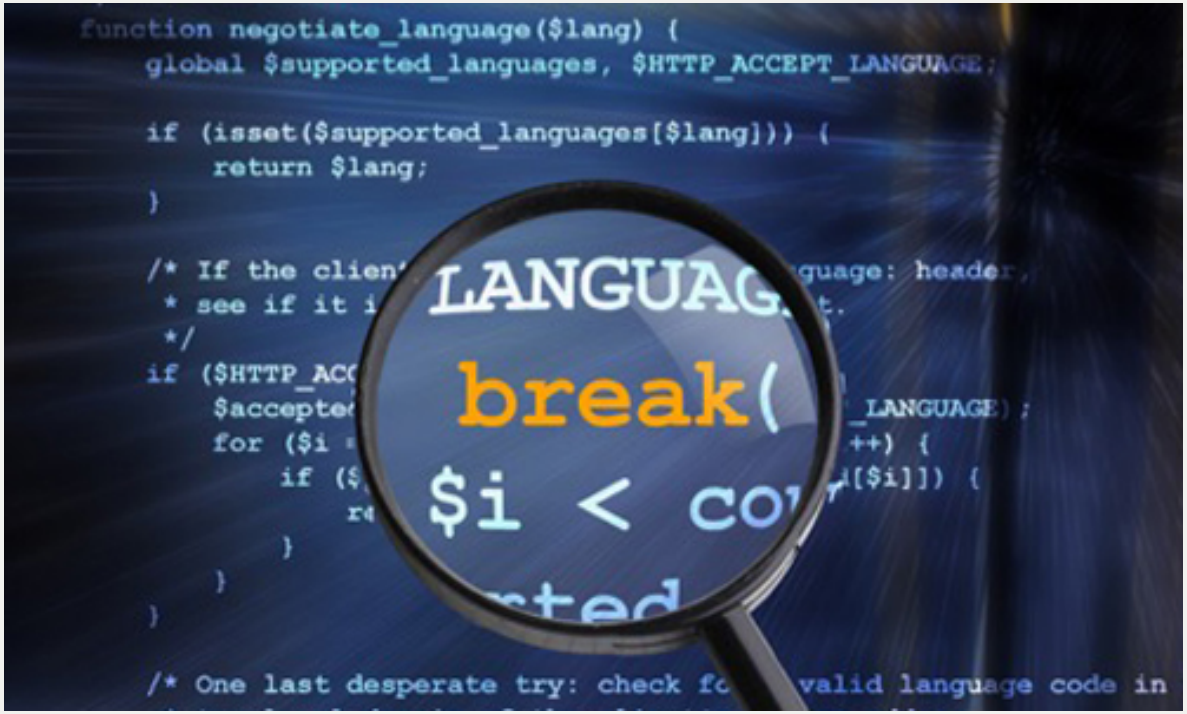
```
public static Material GetFadedMaterial()
{
    if (fadedMaterial == null)
        fadedMaterial = Resources.Load("gui/blueprint faded") as Material;
    return fadedMaterial;
}

public void StartBlueprint(Thing thing)
{
    if (blueprintStructure != null)
        Destroy(blueprintStructure);
    blueprintMode = BlueprintModes.expanding;
```

```
#include<iostream>
using namespace std;
int main()
{
    int armstrong=0,num=0,result=0,check;
    cout<<"Enter a Number to find it armstrong or not";
    cin>>num;
    check=num;
    for(int i=1;num!=0;i++){
        armstrong=num%10;
        num=num/10;
        armstrong=armstrong*armstrong*armstrong;
        result=result+armstrong;
    }
    if(result==check){
    cout<<check<<"  is an Armstrong Number";
    }else{
    cout<<check<<"  is NOT an Armstrong Number";
    }
    return 0;
}
```

Secure code reviews

What do we ask for?

1 Requirements
2 Design
3 Source code
4 Demo
5 Security development process info



Secure code reviews

What do we get?

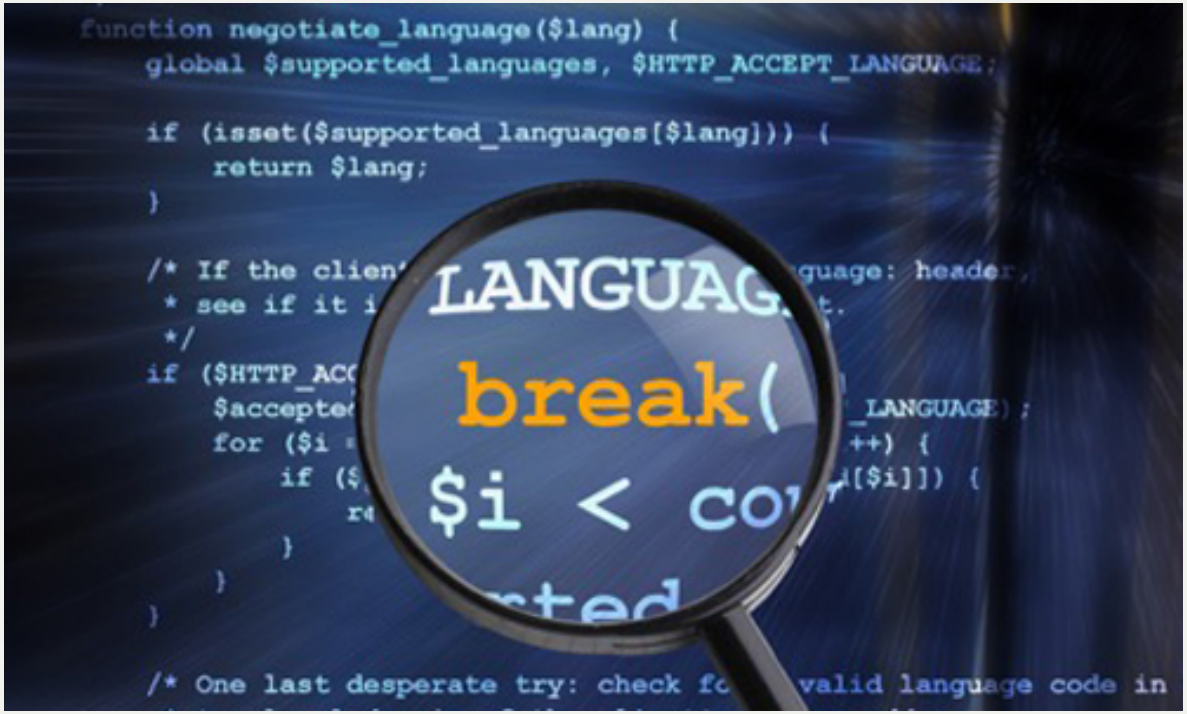
1 ~~Requirements~~

2 ~~Design~~

3 Source code

4 Demo

5 ~~Security~~ development process info



What do we see in the code?

Best practices we often find

1. **Parameterized queries**
   - Mostly because frameworks provide ORM that automatically provide parameterized queries;
2. **Data in the front-end is properly encoded**
   - Mostly because front-end applications rely on frameworks that properly escape users' inputs;
3. **Authentication mechanisms are properly implemented**
   - Sometimes, HTTP filters in the backend have a strange behavior;
4. **Communications with the internet are being protected**
   - However, cipher suites are the default ones
5. **Security frameworks are used**
   - Increased security level with very low effort
6. **Global mechanism for exception handling are implemented**
   - Rarely stack traces or raw exception messages are shown directly to the user

Best practices we rarely find

1. **Missing HTTP headers**
    1. Cache control mechanisms are missing
    2. Secure cookies are not often used, etc.
2. **Access control is not enforced** for all system functions and is rarely centralized
3. **Session** tokens are not random enough or depend on users' data
4. **Log files** do not contain all necessary information to perform auditing or intrusion detection
    - Log files containing sensitive information
5. **Connections in the backend** (internal network) are rarely protected
6. **Hardcode passwords** everywhere (mostly application passwords)
    - Passwords storage in the clear (either in the session or in the users' browser or in the database)
7. **Bad usage of crypto** libraries
8. **Libraries are rarely checked** for vulnerabilities

Access control problems

1 Why so many access control problems?

1. Trend is to perform access control in the frontend
2. Access control is rarely centralized
3. Not considered in the design/requirements definition phases;

2 What happens almost of the times?

- Access control is performed locally for certain system functions, either using:
  1. A framework
  2. Or implementing ad-hoc/local access control mechanisms

Development *versus* Production Code

How to insert backdoors in your system?

Mix up **development** and **production** code and have different configurations for the different environments!

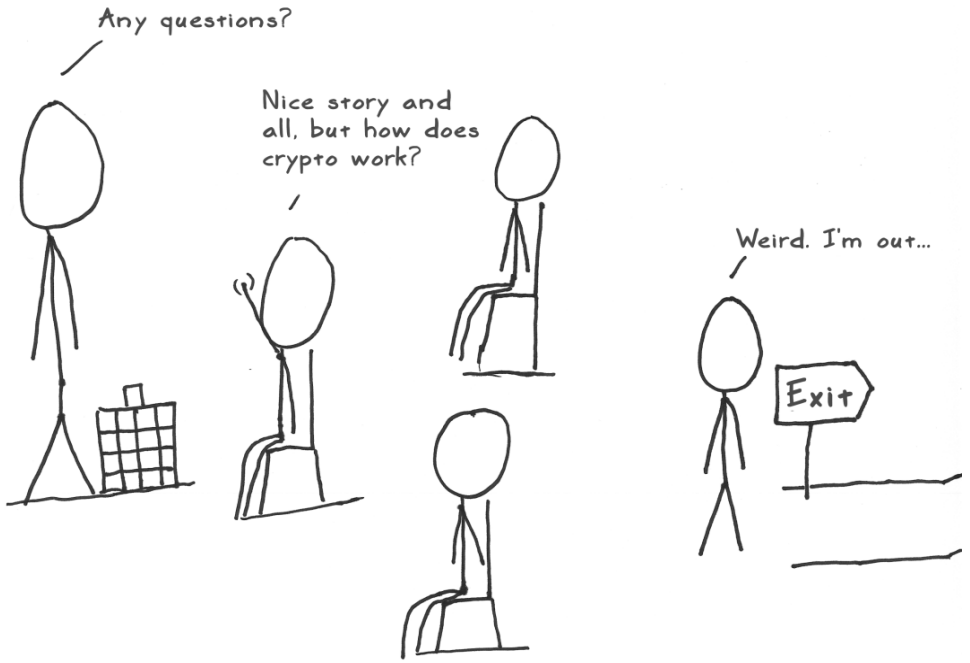
```
def run_user_code(envdir):
    source = raw_input(">>> ")
    try:
        if isDevEnvironment:
            exec source in envdir
        else: pass
    except:
        if isDevEnvironment :
            print traceback.print_exc(file=sys.stdout)
        else: 'File error'
envdir = {}
while 1:
    run_user_code(envdir)
```

Leaner is safer

The Crypto Problem

Already in 2006, Chris Eng presented some common crypto mistakes at Black Hat 2006;

- Proprietary or home-grown encryption algorithms;
- Insecure operation modes (ECB, OFB, etc.);
- Poor key selection;
- Insufficient key length;
- Inappropriate key reuse;
- Insecure random number generation;
- Incorrect use of crypto API;



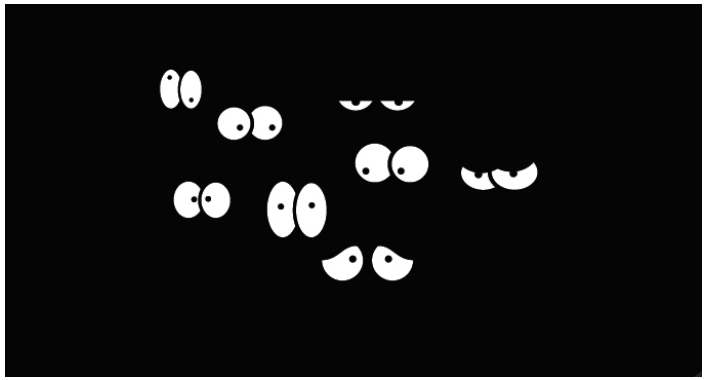
What has changed in 10 years?

1. No more home-grown or proprietary encryption algorithms;
2. Key lengths are now set properly to be higher than 128bits ;

But everything else remains the same...

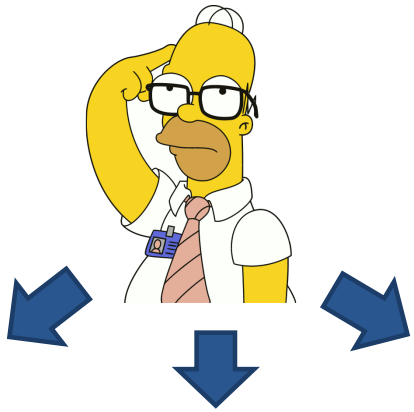
1. Mode of operation is set by the underlying security framework (usually a recommended one);
2. Keys are derived from some hardcoded application key (always the same key);
3. Keys are reused for every encryption;
4. Insecure random number generation;
5. Incorrect use of crypto API mostly due to bad examples;
6. Passwords are hashed with weak algorithms (MD5);
7. ...

Where does the problem lie?



Security is a difficult concept to grasp

Forums full of insecure examples



Security requirements are not properly defined

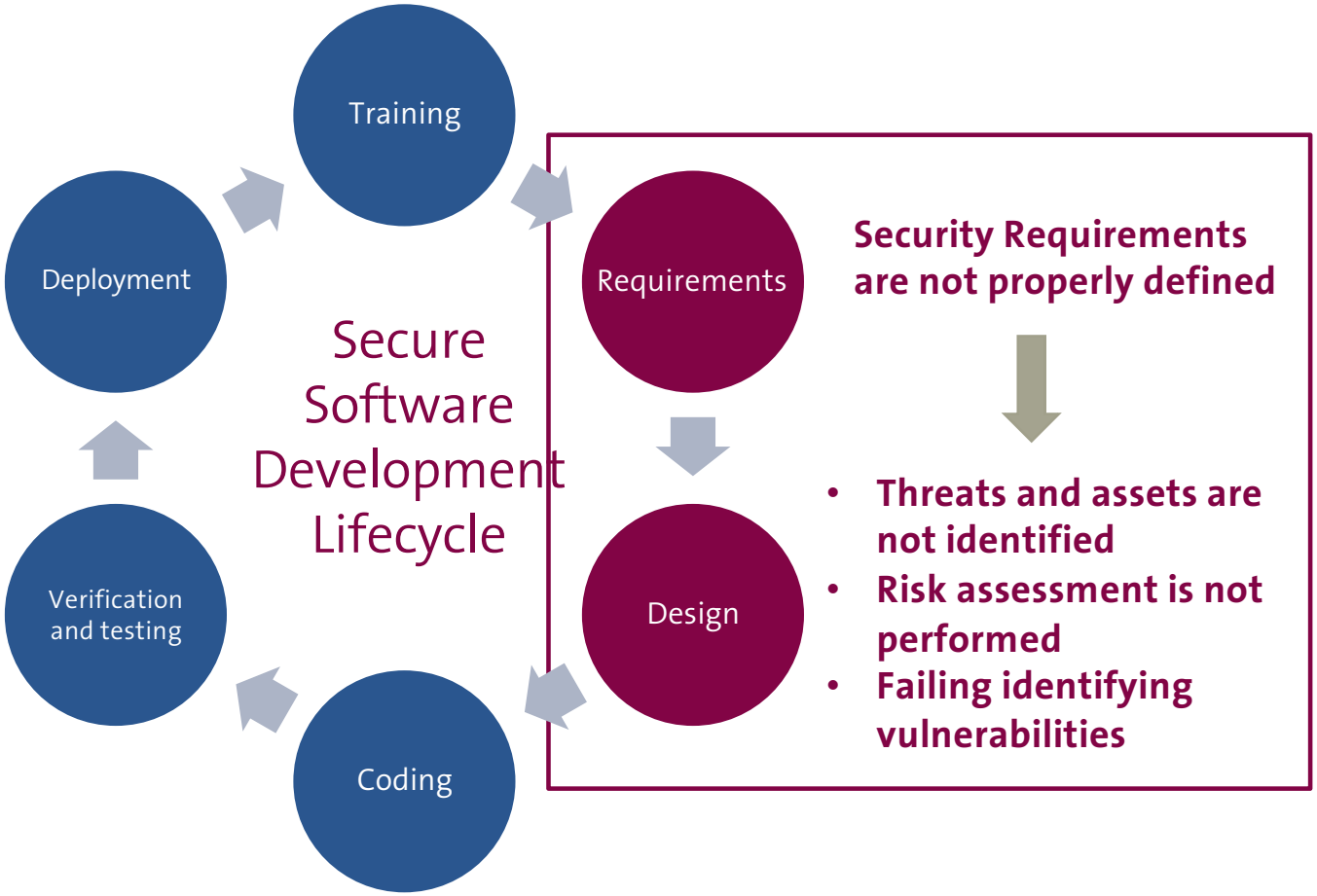
Training is not always effective

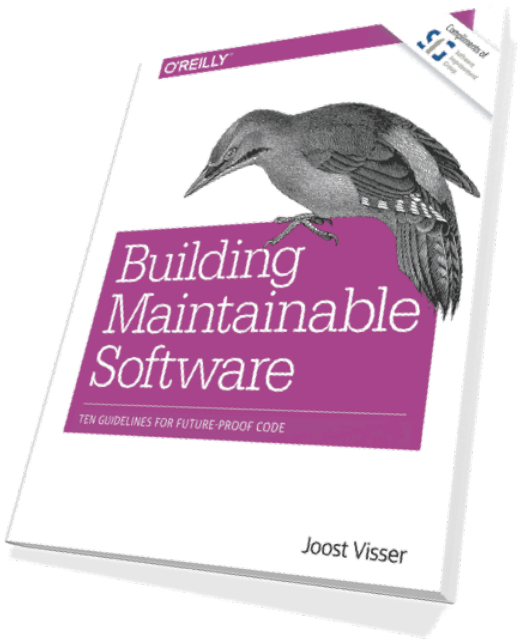
Frameworks and libraries defaults might be insecure

Where does it start?

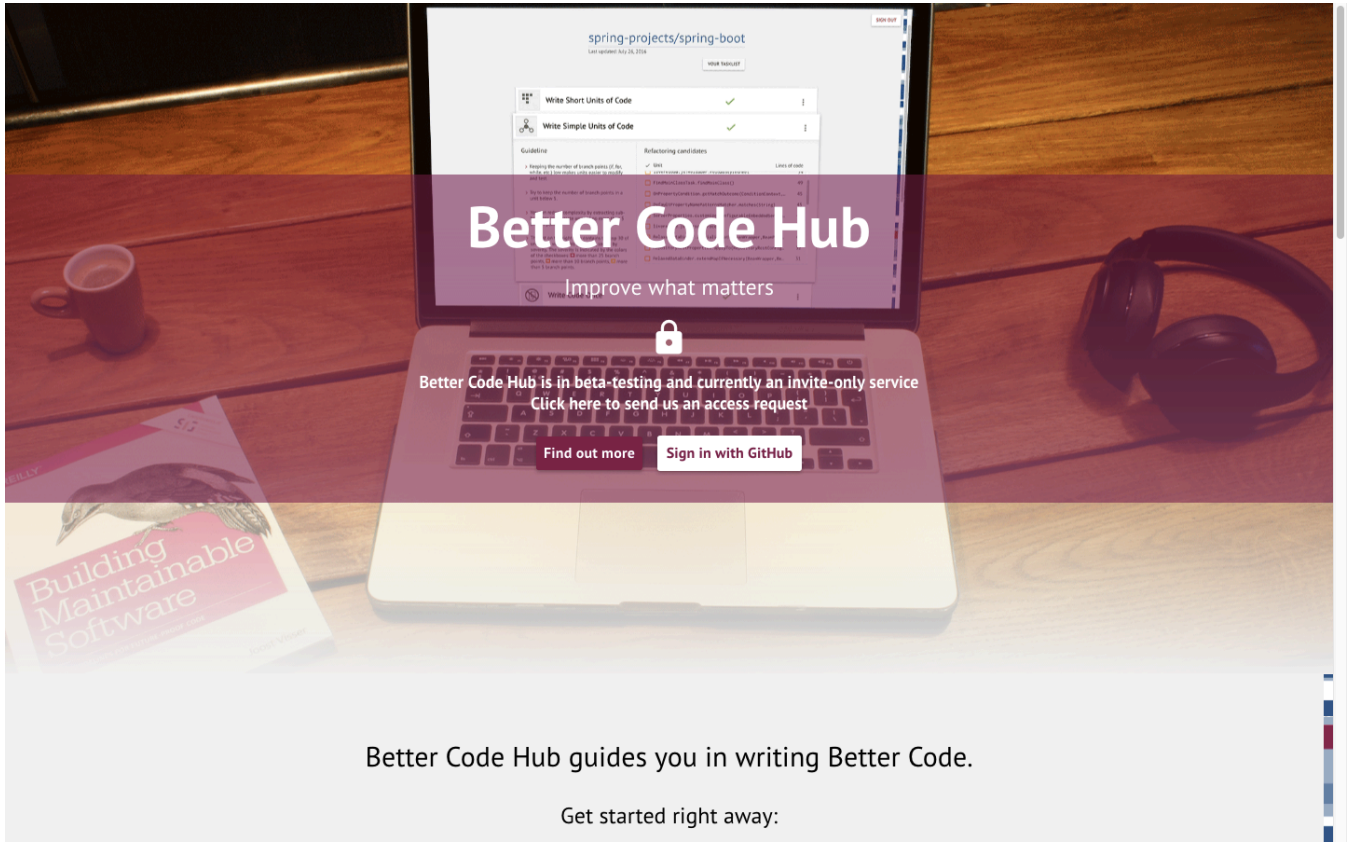
Often Missing:

- Verifying that the requirements are satisfied
- Proper testing and code reviews
- Integration of SAT & DAST





"Maintainability is not an afterthought, but should be addressed from the very beginning of a development project. Every individual contribution counts."



https://bettercodehub.com

Software Improvement Group

Contact

+31 621 530 113

b.vieira@sig.eu

@barbaraisv