Stressful projects?
Become a greater developer
NoBugs Project
Come to
Bad code?
Lots of bugs?
Learn the BEST programming techniques

No Bugs, No Stress

Your Step by Step Guide to Creating Life-Changing Software Without Destroying Your Life

The problem

What are the deepest and real problems regarding bugs?

What developer has never had real problems regarding bugs?

Do you know where the company spends more money?

If you were thinking about software maintenance, you are completely right!

Again and again many companies commits the same mistake, they don't care about software quality.

What happens if nobody cares about software quality?

Your life-quality becomes worse and worse, you won’t have time for you and your family.

You won’t improve your programming skills as much as you could.

Your health will be also affected because when you work overtime, probably you won’t have healthy eating.

You will be focused on all the stress you are having in your job!

They can truly ruin our lives!

But…. Fortunately, there is a way to deal and solve this problem!

The solution is in this **book**! Read it with attention and find out how!

Step 1 – Learn the Best Programming Practices

1.1 - Stop a moment to think about the name of the package/class/method/variable

YES, names are **EXTREMELY** important.

With a self-describing name, it is very easy to find out what your code is doing.

For this reason, do not use the first name that pops into your head. Think about the best possible name that describes exactly for what your package, class, method, variable is responsible.

Avoid ambiguous names! If a bug occurs in this functionality, it will be very difficult to find it.

Examples of ambiguous names:

```
Double clientValue; // Value of the client, value of what?
void hide() // What does that hide?
class Historic {} // Historic of what? Historic for everything? Where is the
cohesion?
```

Specific names help A LOT!

1.2 – Get to know the shortcuts of your IDE

I am going to use Eclipse’s example. It’s very important to use the shortcuts so you can code two or three times faster.

Control + 3 = “Eclipse’s Google” – write “new class” for example.

Control + shift + F = format your code.

Control + 1 = do various actions. For example, invoke a method that does not exist yet and use the shortcut.

Configure Eclipse’s formatter. Change it from 80 characters per line to 120. Configure save actions too so you can have actions when you save your code on Eclipse. For example, we can use the command “organize imports” after each save.

There are many other shortcuts, but remember, you need to use them until they get automatic for you. If you forgot to use the shortcut, delete the code and use the shortcut, so you'll never forget it.

More sources:

Eclipse Shortcuts

Netbeans Shortcuts

IntelliJ Shortcuts

1.3 – Encapsulate methods for what you need

Have you noticed a lot of code in your application that does the same thing?

Why not encapsulate a method for this?

For example:

```
Calendar c = Calendar.getInstance();
c.setTime(new Date());
c.add(Calendar.DATE, +1);
Date nextDay = c.getTime();
```

It would be a lot easier and much clearer to encapsulate all this code in a separated class and method:

```
Date nextDay = DateUtils.getNextDay();
```

Consider if need to keep state of your object, if so, instantiate it!

Remember, the most important thing here, is to **encapsulate your solution**!

1.4 – Create the culture of Code Review in your company

Even if the company you work for doesn’t use a Code Review tool, ask your friend to take a look at your code.

Each developer has his own experiences. We can learn a lot by sharing them.

Many things that you may not have noticed in a simple Code Review could be uncovered in a peer Code Review. You will see your mistakes and learn from them!

Step 2 – Master Business Rules

2.1 – Be a user, use the system

We are developers, most times, we say bad things about users. But the truth is, we are users too! It's a big mistake to think we should not use the system. How are you going to understand the process? Reading the code? It can be possible, but it will take time more time for doing it!

So, use the system, see what happens, and check how the information is changed.

Read the menu of the system, navigate, do some actions, insert, update, delete, search information!

See what happens on the main flows of the system!

2.2 – Test queries based on the business rules

Do you know the main **Database Tables** from the project you are working on? What happens if you don't know the **Tables**? You won't be able to use your creativity and use the best solution. Maybe you will do unnecessary things to achieve your goal. You will use a lot of unnecessary **Join** commands and the chance of getting **bugs** will be very high! Why? If you don't know what you are doing exactly, how can you guarantee you are not making a mistake?

You won't. Before you have unnecessary stress, just test **Queries**, see how they work with main business rules. Use **Join**, **Group by**, **Left Join**, **Union**, everything you know to test and understand what is happening.

You can grab these queries on the main services of the project, get and test them!

2.3 – Talk to Solution/Business people

If you are working with **Solution/Business** people, ask them!

Mainly, if you work with **SCRUM**, one of its principles is to work together with all members of the team. Everyone has to help each other to achieve the goal. If you are not working with **SCRUM**, no problem, even so, ask them to get your answer!

If you don't understand the system as a whole, you will have **bugs** and unnecessary stress.

2.4 – Create a Wiki (internal site)

Wiki! Create a **Wiki**! Very important! How new developers will know about the business rules? They will need to ask about everything to other developers?

It shouldn’t be the solution. The business rules should be in an easy and central place, so that new developers would be able to learn and understand what is happening.

If you have a **Wiki**, new and more experienced developers in the company will produce more because they won’t be obligated to stop and explain everything.

The company will earn more money for sure!

Ask your team members to write the core business rules, database and so forth!

You can’t learn business rules at home, so we must have a source of information to learn them in an easier way!

2.5 – Read the interfaces (API)

Read the interfaces (API) of the project you are working on, read the comments (if present) or simply the method signature

If you don’t have **Business/Solution** people on your project or developers don’t want or don’t have time to explain how the business rules work, you must read the code.

I know it’s very complicated when the code was written using Go Horse process but it’s the only option.

If the code is unreadable, just see how the query works, try to identify the most important **Database Tables** from the system you are working on.

The most important thing is, don’t stop your work! Make your way to learn the business rules using all of these techniques.

Don’t keep on waiting for a magic solution to happen, you must use these techniques and solve the problems!

Step 3 – Be persuasive, know how to negotiate

3.1 – Learn how to say no

What developer has never worked with short deadlines?

I believe most of us have already had this experience and, as you know, it's extremely stressful!

Sometimes managers want to show numbers instead of quality.

So what do we have to do? We have to say no! We can't accept short deadlines.

Provide clear reasons against short deadlines – unnecessary stress, poor quality, and an increase of bugs!

In the end, shorter deadlines can result in greater expenses for the company including overtime for developers and loss of credibility with clients.

If you don't say no to short deadlines:

Master Yoda would say:

Your project, in chaos, will be. Multiply, bugs will.

3.2 – Break up your task into sub-tasks

If your task seems too broad, you have to break it up!

It is impossible to know how much time you will take to deliver a task if you don’t know exactly what it is.

If you don’t know the business rules, the database model, or the technology you will use, for sure you will take more time just to understand what you have to do.

Remember, even breaking up the tasks will take time.

3.3 – Use a development methodology (SCRUM, XP, LEAN, KANBAN, etc)

Yes, use a development methodology. Using a methodology is much better than not.

There are a lot of very good methodologies like **SCRUM**, Extreme Programming, Lean and Kanban. You can mix and match parts of different methodologies and adapt them to your project.

For example, you can use **KANBAN** with **SCRUM** and use pair programming from the **Extreme Programming** methodology.

Pair programming is very useful. It can balance the team experience and developers can learn from each other. For complex tasks, it’s extremely useful. If you have never tried, when you do, you will think the same!

It’s easier to control software development with a methodology.

You can define how much you can deliver on your “sprint”. Most times the sprint lasts 2 weeks.

3.4 – Use complex points! (Define the complexity of your task)

At first break up all of your tasks so your team is capable of understanding what must be done.

Once you have organized your tasks, you will assign a point to each task.

On SCRUM methodology, we use Fibonacci numbers, 1, 2, 3, 5, 8, 13,

21, 34, 55…

These numbers are very subjective. You can define your own pattern, but typically, it is like this:

- **1 to 3 – Easy**
- **3 to 8 – Medium**
- **13 to … – Hard**

You must consider everything when you are defining complex points –

business rules, database model and technology.

3.5 – Be persuasive and convince everyone that you won’t have enough time to deliver the product

Do you think you have to be just a technical person? No! It’s one of our mistakes. We must develop other skills too! How can you convince your manager or your client if you don’t practice persuasion skills? Learn how to negotiate and save your project!

Books to read for learning how to be persuasive:

- **Getting More**, How You Can Negotiate to Succeed in Work and Life, Stuart Diamond
- **Crucial Conversations**, Tools for Talking When Stakes Are High Kerry Patterson, Joseph Grenny, Ron McMillan, and Al Switzler
- **Influence**, The Psychology of Persuasion, Robert B. Cialdini

If you don't have enough time to read the books, just think about how a short deadline can damage the project and explain it.

Make them understand they are making a mistake!

Step 4 – Test your software

4.1 – Test business rules from your methods

Well, what should be tested? Business rules, for sure! Don’t implement unitary tests to cover all methods and create numbers, make a real test!

Know your method and test it with meaningful flows where you can guarantee your logic is working.

Don’t implement meaningless tests.

4.2 – Give a clear, meaningful name to your method

Think about a great name for your test and choose the name that represents what you are testing.

Bad example:

```
@Test
public void test1()
```

Good example:

```
@Test
public void checkIfCustomerIsBlockedByFraudTest()
```

Much easier to understand, huh? You know your method is checking if the customer is blocked by fraud.

4.3 – If your method is too complex, use TDD (Test Driven Development)

Have you ever had to implement a very complex method? If so, it’s very difficult to test, right? To make implementation easier you should use **TDD** and implement your test before implementing your method. Use baby-steps, do the simplest thing first and then make your test work! Then keep going until your method is completely implemented!

The cool thing about using TDD is that your code is already tested! It’s amazing how it can help you! Just practice it, and see how amazing this is!

4.4 – Create your classes/methods with cohesion

Unitary tests help a lot to know if your code is good or bad. It’s very simple, if you can’t test your code easily, your code doesn’t have cohesion or low coupling.

If you are in this situation, refactor your code, don’t turn it into a big problem. If your code keeps on growing like this, you will have real problems to change features or implement something new.

If you implement a test for a method that is responsible for many actions, it will be completely useless. Anything you change on your method can break the test.

Avoid unnecessary stress by just refactoring your code.

4.5 – Know the difference between Unitary Tests and Integration Tests

What is the difference? Easy! Unitary Tests evaluate just one method. They test your code unitarily, method by method. You won’t use information from the database. You will “Mock” data, meaning you will create your own “fake” data to implement the test you need for your business rule. Although you can’t interact with the database, Unitary Tests are very fast and it’s possible to test business rules effectively.

Integrated tests, as the name says, are more complete tests. You will interact with the database. You will insert your own data into the database and test your services integrally. The cons are that it is slow, it takes time to test everything. You also take more time to implement the test. It’s necessary to create your own data directly in the database (insert script, delete script).