There Will Be Code:

- We will never be rid of code, because code represents the details of the requirements. At some level those details cannot be ignored or abstracted; they have to be specified. And specifying requirements in such detail that a machine can execute them is programming. Such a specification is code.

- the author expect that the number of domain specific languages will continue to grow. This will be a good thing. But it will not eliminate code.

- -if the discipline of requirements specification has taught us anything, it is that well-specified requirements are as formal as code and can act as executable tests of that code!

- Remember that code is really the language in which we ultimately express the requirements.

==========

Bad Code

reasons for writing bad code:

- Were you trying to go fast?.

- Were you in a rush?

- Perhaps you felt that you didn’t have time to do a good job; that your boss would be angry with you if you took the time to clean up your code.

- Perhaps you were just tired of working on this program and wanted it to be over.

- Or maybe you looked at the backlog of other stuff that you had promised to get done and realized that you needed to slam this module together so you could move on to the next.

- LeBlanc’s law: Later equals never.

==========

The Total Cost of Owning a Mess

- teams that were moving very fast at the beginning of a project can find themselves moving at a snail’s pace.

- As the mess builds, the productivity of the team continues to decrease, asymptotically approaching zero. As productivity decreases, management does the only thing they can; they add more staff to the project in hopes of increasing productivity. But that new staff is not versed in the design of the system. They don’t know the difference between a change that matches the design intent and a change that thwarts the design intent. Furthermore, they, and everyone else on the team, are under horrific pressure to increase productivity. So they all make more and more messes, driving the productivity ever further toward zero.

==========

Attitude

- The managers and marketers look to us for the information they need to make promises and commitments; and even when they don’t look to us, we should not be shy about telling them what we think. The users look to us to validate the way the requirements will fit into the system. The project managers look to us to help work out the schedule. We are deeply complicit in the planning of the project and share a great deal of the responsibility for any failures; especially if those failures have to do with bad code!

- it is unprofessional for programmers to bend to the will of managers who don’t understand the risks of making messes.

==========

The Art of Clean Code?

- The only way to make the deadline—the only way to go fast—is to keep the code as clean as possible at all times.

- being able to recognize good art from bad does not mean that we know how to paint. So too being able to recognize clean code from dirty code does not mean that we know how to write clean code!

- “code-sense” is the key. Not only does it let us see whether code is good or bad, but it also shows us the strategy for applying our disci- pline to transform bad code into clean code.

- a programmer who writes clean code is an artist who can take a blank screen through a series of transformations until it is an elegantly coded system.

==========

What Is Clean Code?

Bjarne Stroustrup, inventor of C++ and author of The C++ Programming Language:

"I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well."

- clean code is pleasing to read. Reading it should make you smile the way a well-crafted music box or well-designed car would.

- Bad code tempts the mess to grow! When others change bad code, they tend to make it worse.

- Clean code is focused. Each function, each class, each module exposes a single-minded attitude that remains entirely undistracted, and unpolluted, by the surrounding details.

==========

Grady Booch, author of Object Oriented Analysis and Design with Applications

"Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of crisp abstractions and straightforward lines of control."

==========

“Big” Dave Thomas, founder of OTI, godfather of the Eclipse strategy

"Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone."

- There is a difference between code that is easy to read and code that is easy to change.
- Code, without tests, is not clean. No matter how elegant it is, no matter how readable and acces- sible, if it hath not tests, it be unclean.
- Smaller is better.

==========

Michael Feathers, author of Working Effectively with Legacy Code

"I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. Clean code always looks like it was written by someone who cares. There is nothing obvious that you can do to make it better. All of those things were thought about by the code’s author, and if you try to imagine improvements, you’re led back to where you are, sitting in appreciation of the code someone left for you—code left by someone who cares deeply about the craft."

==========

Ron Jeffries, author of Extreme Programming Installed and Extreme Programming Adventures in C#

"In recent years I begin, and nearly end, with Beck’s rules of simple code. In priority order, simple code:
• Runs all the tests;
• Contains no duplication;
• Expresses all the design ideas that are in the system;
• Minimizes the number of entities such as classes,methods, functions, and the like."

==========

Ward Cunningham, inventor of Wiki, inventor of Fit, coinventor of eXtreme Programming. Motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code.

"You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the code also makes it look like the language was made for the problem."

- We’ve all railed against the fact that our languages weren’t designed for our problems. But Ward’s statement puts the onus back on us. He says that beautiful code makes the language look like it was made for the problem! So it’s our responsibility to make the language look simple! Language bigots everywhere, beware! It is not the language that makes programs appear simple. It is the programmer that make the language appear simple!

==========

We Are Authors

- We are authors. And one thing about authors is that they have readers. Indeed, authors are responsible for communicating well with their readers. The next time you write a line of code, remember you are an author, writing for readers who will judge your effort.

- We are constantly reading old code as part of the effort to write new code.

- we want the reading of code to be easy, even if it makes the writing harder. Of course there's no way to write code without reading it, so making it easy to read actually makes it easier to write.

- You cannot write code if you cannot read the surrounding code. The code you are trying to write today will be hard or easy to write depending on how hard or easy the surrounding code is to read. So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

==========

The Boy Scout Rule

- It's not enough to write the code well. The code has to be kept clean over time. We've all seen code rot and degrade as time passes. So we must take an active role in preventing this degradation.

- If we all checked-in our code a little cleaner than when we checked it out, the code simply could not rot.

- The cleanup doesn't have to be something big

==========

Conclusion

In many ways this book is a "prequel" to a book I wrote in 2002 entitled Agile Software Development: Principles, Patterns, and Practices (PPP). The PPP book concerns itself with the principles of object-oriented design, and many of the practices used by professional developers. If you have not read PPP, then you may find that it continues the story told by this book. If you have already read it, then you'll find many of the sentiments of that book echoed in this one at the level of code.