Simple, Simplest, or Simplistic?

ACCU 2006 Spring Conference
Giovanni Asproni
gasproni@asprotunity.com

1

It is not about “Agile”

2

Even if the idea for this presentation comes from the Extreme Programming motto “do the simplest thing that could possibly work”, the content applies to any process: the underlying assumption is that nobody wants to do unnecessary complex things. Even “Big Design Upfront” people do that because they think that gives a simpler solution.

It’s about quality

3

In this talk we are interested in *simple code* as a way to produce bug-free and maintainable software faster and more cheaply.

Nothing new actually…

“None of the ideas presented here are new; they are just forgotten from time to time.”

Alan J. Perlis, 1966 Turing Award Lecture.

Summary

- Some definitions
- Why care?
- Simplicity
- Simplistic software
- Solutions?

Simple is not simplistic…

"Very often, people confuse simple with simplistic. The nuance is lost on most."

Clement Mok, Chief Creative Officer, Sapient.

From “The New Oxford Dictionary Of English”

- **Simple**: *easily understood or done; presenting no difficulty*
- **Simplistic**: *treating complex issues and problems as if they were much simpler than they really are*
- ***Complex****: consisting of many different and connected parts. not easy to understand; complicated.*

8

Example: simple vs. simplistic

```
typedef std::set< Foo > SetOfFoo;

…

void doSomething( SetOfFoo& foos );

…

void getSomething( const SetOfFoo& input,
                   SetOfFoo& result );
```

9

This C++ example comes from experience in real projects. It is simple in a context in which a set of foos makes sense in the domain, and the std::set class has an interface that is sufficient for the needs of the application. Giving a name to a concept makes it easier to look for it in code, and to change its implementation if necessary, touching only a small part of the code-base. Of course, the fact that C++ typedefs are just aliases can cause trouble if the programmers are not disciplined enough, and use std::set< Foo > directly instead of the typedef.

Example: simple vs. simplistic

```
/**
 * foos doesn’t have any duplicates…
 */
void doSomething( std::vector< Foo >& foos );

…

/**
 * input doesn’t have any duplicates…
 * result doesn’t have any duplicates…
 */
void getSomething( const std::vector< Foo >& input,
                   std::vector< Foo >& result );
```

10

This C++ fragment is an obvious simplistic solution. Two major problems are the usage of std::vector instead of std::set, and the lack of typedef. This code is error prone, and a potential cause of lots of duplication to check that the vector is actually a set.

Simplicity depends on context

“Everything should be made as simple as possible, but not simpler.”

Albert Einstein.

Simplicity depends on context

A simple solution to a complex problem is a solution whose complexity matches the complexity of the problem itself

12

Processing a file line by line in Java 1/2

```
public class FileProcessor {

    public static void main( String[ ] args ) {
         try {
             BufferedReader reader = new
             BufferedReader( new
                 FileReader( “infilename“ ) );

              while ( true ) {
                 String line = reader.readLine( );
                 if ( line == null )
                     break;
                 processLine( line );
             }
             reader.close( );
         }
         catch ( Exception exc ) { exc.printStackTrace( ); }
    }
…
}
```

13

This can be seen as an example of simple software in a context in which we want to process a file in some way, and then dispose of the code. In this case we are not really interested in managing the exceptions that may be raised.

Processing a file line by line in Java 2/2

```
public class FileProcessor {

    public void process( String fileName ) {
         try {
             BufferedReader reader = new
             BufferedReader( new
                 FileReader( fileName ) );

              while ( true ) {
                 String line = reader.readLine( );
                 if ( line == null )
                     break;
                 processLine( line );
             }
             reader.close( );
         }
         catch ( Exception exc ) { exc.printStackTrace( ); }
    }
…
}
```

14

This is what happens if we “adapt” in a simplistic way the code from the previous slide to work in a more general context. In this case, the solution is a simplistic one, because the exceptions are not managed properly, and the function cannot be tested easily. Try to imagine what would happen in a server application supposed to run 24x7.

Why care? 1/2

- We often talk about over-engineering
  - Traditional processes vs. agile processes
  - Big design up-front vs. continuous design
  - Etc.

Why care? 2/2

- A lot of software out there is under-engineered, i.e., *simplistic*
- Simplistic solutions lead to unnecessary complex software that is more buggy, less maintainable, less extendable, etc.

16

Some software is over-engineered, and under-engineered at the same time: it tries to solve several problems too far ahead in the future, each of them in a simplistic way.

Simplicity is at the very foundation of Software Engineering…

"I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.
The first method is far more difficult…"

C. A. R. Hoare, "The emperor's old clothes."

Simplicity is at the very foundation of Software Engineering…

- Decomposition
- Abstraction
- Layering
- Encapsulation
- Information hiding
- Etc.

18

Software Engineering principles are usually presented as a way to attack complexity, but they can be seen as a way to strive for simplicity.

…Because simple software is

- Maintainable
- Reliable
- Testable
- Extendable
- Readable
- Usable
- Of higher quality

19

Simplicity depends on perspective

Different people have different views

Simple software is…

- …For developers
  - Maintainable
  - Reliable
  - Testable
  - Extendable
  - Readable
  - Etc.
- Internal quality

21

Simple software is…

- According to Kent Beck:
  - Appropriate for the intended audience
    - The people that have to work with it have to understand it
  - Communicative
    - Every idea that needs to be communicated is represented in the system
  - Factored
    - No duplication of logic or structure
  - Minimal
    - The system should have the fewest elements possible

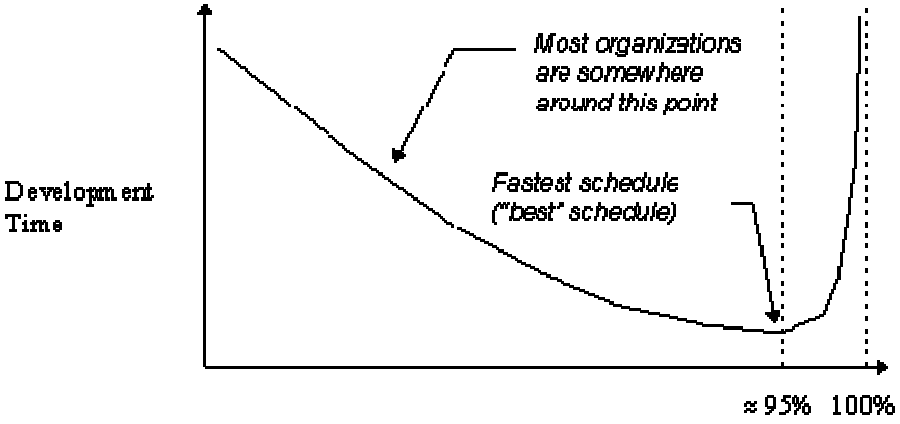
22

Simple software is…

- …For users:
  - Usable
  - Simple to learn
  - Well documented
  - Reliable
  - Fit for purpose
  - Fast
  - Etc.
- External quality

Internal quality affects external quality

Relationship between quality and development time



Source: Steve McConnell. http://www.stevemcconnell.com/articles/art04.htm

Types of complexity

- Fred Brooks in “The Mythical Man Month” defines complexity as
  - Essential
    - Inherent to the nature of the problem
  - Accidental
    - Limitations in tools, knowledge, etc.

26

These definitions are necessary to understand where simplistic solutions come into play.

Example: essential vs. accidental complexity

- Essential complexity
  - A text file editor has to support create/open, close, save, and edit functionality for text files
- Accidental complexity
  - Configuration management
  - Software builds
  - Testing
  - Programming languages, IDEs, etc.

27

Example: processing a file line by line in Java

```
public class FileProcessor {

    public static void main( String[ ] args ) {
         try {
             BufferedReader reader = new
             BufferedReader( new
                 FileReader( “infilename“ ) );

              while ( true ) {
                 String line = reader.readLine( );
                 if ( line == null )
                     break;
                 processLine( line );
             }
             reader.close( );
         }
         catch ( Exception exc ) { exc.printStackTrace( ); }
    }
…
}
```

28

Example of accidental complexity introduced by a programming language—Java requires a lot of boiler-plate code.

Example: processing a file line by line in Python

```
inFile = open( “infile“ )
for line in inFile :
    processLine( line )
inFile.close( )
```

29

Example of accidental complexity introduced by a programming language—Python, in this case, introduces less complexity than Java.

Accidental complexity

- A certain amount of it is **unavoidable**
  - Configuration management
  - Build scripts
  - Languages and libraries
  - Etc.
- Simplistic solutions introduce **avoidable** accidental complexity

Example: avoidable complexity

- Same class used to model distinct hierarchies
  - Location (continent, country, town)
  - Organizational unit (Resources, Human Resources, Contractors, IT Contractors)
  - Both modelled with same Java class “Node”
  - Both mapped into “node” database table

31

This is an example drawn from a real Java project. A class Node was used to model to distinct hierarchies, and a corresponding “node” table was used to store them in the database. In the code there was no obvious way to distinguish among them, except by knowing which classes generated the data. This particular class was the cause of some nasty bugs and test headhaches.

Simplistic solutions: the official reasons…

- "We have no time now…"
  - "…will fix it later"
  - "…for writing tests"
- "The users really need this functionality now…"
- "If we don't deliver this functionality now the business will lose X million pounds…"
- "…but we are different…"
- "…but in the real world..."

32

In my experience, these are very common reasons given to justify simplistic solutions. However, they tend to be given without really fully appreciating the problems at hand. The last one is especially bad, since it is usually given in a patronizing way implying that the other person lives in some kind of "ideal world" where things are obviously different from the real one. It is often used to shut down any discussion and thinking as well.

…And some real ones

- No clear-cut definition of what makes software simple
- Wishful thinking
  - Technology and tools
  - Development processes
- Fear
- Social issues
- Lack of knowledge
  - No learning from previous experience

33

These are, in my experience some real reasons for choosing simplistic solutions.

As a result the simplest solution...

…Is often defined as the *quickest to hack*

34

No clear-cut definition of what makes software simple

- Is an elusive concept
  - How can we define it?
  - How can we measure it?
  - How can we recognize simple software?
- Only heuristics and rules of thumb

35

Some people define metrics to measure software complexity—McCabe, Robert Martin, Halstead—however they are not very satisfactory. They come “after the fact”, so they can give only limited guidance—by knowing what the average numbers of the code-base are, they can point-out some places where the numbers look strange, but cannot give any real guidance on how to make the code simpler.

Wishful thinking: technology and tools 1/7

- XML will solve all our data distribution problems
  - Data previously distributed in ASCII files
  - Half terabyte dataset
  - XML would increase the size by an order of magnitude
  - Users wanted to edit files using emacs

```
ID   BP195628   standard; mRNA; EST; 457 BP.
XX
AC   BP195628;
XX
SV   BP195628.2
XX
DT   09-SEP-2004 (Rel. 81, Created)
DT   11-MAR-2006 (Rel. 87, Last updated, Version 5)
XX
DE   Homo sapiens cDNA clone: ADB08718, Sugano cDNA library, expressed in brain,
DE   5'-EST.
XX
KW   5'-end sequence (5'-EST); EST (expressed sequence tag).
XX
OS   Homo sapiens (human)
OC   Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia;
OC   Eutheria; Euarchontoglires; Primates; Catarrhini; Hominidae; Homo.
XX
RN   [1]
RP   1-457
RA   Suzuki Y., Yamashita R., Shirota M., Sakakibara Y., Chiba J., Nakai K.,
RA   Sugano S.;
…
```

37

A fragment in the original format…

```
<?xml version="1.0" encoding="UTF-8" ?>
…
- <entry accession="BP195628" name="BP195628" division="EST" created="2004-09-09" lastUpdated="2006-03-11" releaseCreated="81" releaseLastUpdated="87"
version="5">
  <description>Homo sapiens cDNA clone: ADB08718, Sugano cDNA library, expressed in brain, 5'-EST.</description>
  <keyword>5'-end sequence (5'-EST)</keyword>
  <keyword>EST (expressed sequence tag)</keyword>
- <reference>
- <citation id="1" type="submission" date="2003-05-20">
  <author>Suzuki Y.</author>
  <author>Yamashita R.</author>
  <author>Shirota M.</author>
  <author>Sakakibara Y.</author>
  <author>Chiba J.</author>
  <author>Nakai K.</author>
  <author>Sugano S.</author>
  <locator>Yutaka Suzuki, The Institute of Medical Science, The University of Tokyo; Minatoku Shirokanedai 4-6-1, Tokyo, Tokyo 108-8639, Japan
          (E-mail:ysuzuki@hgc.jp, Tel:81-3-5449-5343, Fax:81-3-5449-5416) </locator>
  </citation>
  <refPosition begin="1" end="457" />
  </reference>
- <feature name="source">
- <organism>
- <nameset>
  <scientificName>Homo sapiens</scientificName>
  <preferredCommonName>human</preferredCommonName>
  </nameset>
  <taxId>9606</taxId>
- <lineage>
  <taxon>Eukaryota</taxon>
  <taxon>Metazoa</taxon>
  <taxon>Chordata</taxon>
  <taxon>Craniata</taxon>
  <taxon>Vertebrata</taxon>
  <taxon>Euteleostomi</taxon>
  <taxon>Mammalia</taxon>
  <taxon>Eutheria</taxon>
  <taxon>Euarchontoglires</taxon>
  <taxon>Primates</taxon>
  <taxon>Catarrhini</taxon>
  <taxon>Hominidae</taxon>
  <taxon>Homo</taxon>
  </lineage>
  </organism>
```


…and the same fragment in the new XML one.

Wishful thinking: technology and tools 4/7

- Rewrite the entire application in Java, because it is simpler than C++
  - There is a C++ application already in production
  - The application is still evolving

39

This is a typical scenario when dealing with legacy systems. Rewriting the entire application is a typical choice, unfortunately it is also a simplistic one. In fact, since the application is still evolving, the new features need to be put both in the old, and in the new code-base with a duplication of effort. Refactoring the old code base would be a much better choice, unless it is a very small one.

Wishful thinking: technology and tools 5/7

- “Seamless” object-relational mapping
  - Touted advantage: the tool will take care of everything
  - Reality: the tool has limitations that need to be addressed

40

Everything “seamless” in Software Engineering, seems to have the potential to be dangerous. Object-relational mapping is no exception. In a project I was involved, the usage of an object-relational mapping tool, and the lack of an appropriate design to deal cleanly with persistency created very big problems, including very long access times for small datasets in a tiny database.

Wishful thinking: development process 6/7

- Switch to an {agile | formal | iterative | …} process to be more productive
  - Changing process is never easy
  - Lots of social problems to address
  - Requirements are difficult to discover no matter the process

Wishful thinking: processes 7/7

- Using Test Driven Development (TDD) code quality will improve automatically
  - TDD is just a tool
  - It cannot, by itself, improve the code
- Some smells
  - Test of “sunny day” scenarios only
  - Very long test set-ups
  - Weird implementation specific methods in high level interfaces

42

TDD is a just tool. If not used properly, the quality of the produced code can still be quite poor. I have been involved in a project in which the entire code-base was written using TDD, but its quality was abysmal.

Fear

- Time pressure
- Peer pressure
- Manager pressure
- Job security
- End of year bonus

43

Fear leads to cutting corner, and give simplistic solutions to problems.

Social issues

- Event driven management
  - Hitting a moving target
- “Political” games
  - Vaporware is more important than delivering working software
- Lack of self-motivation
  - Boredom
- Wrong incentives
  - Rewarding “safe” decisions

44

All the above are typical reasons for simplistic solutions.

Lack of knowledge: tools 1/5

- C++ has a `std::set` class
  - Everybody knows about `std::map`, `std::vector`, and `std::list`
  - Somehow `std::set` is often forgotten
- The same applies to Java

45

Lack of knowledge leads people to use the wrong tool for the task.

Lack of knowledge: C++ forward declarations 2/5

- In a C++ project I was involved
  - The usage of include directives inside headers was forbidden
  - Everything was forward declared, even standard library classes

46

The solution above was clearly a simplistic one. The intent was to reduce compilation times, but the net result was a less portable code-base, and longer compilation times. This second problem happened because it became very difficult to know what headers to include in the files to be compiled, so someone decided to create an header that included all the other headers in the code-base, and to use it everywhere.

Lack of knowledge: design skills 3/5

- Lack of design
  - Lack of domain design (Kevlin's concretion of implied objects)
    - Using strings to represent dates, types, etc.
    - Using signed numbers to represent positive quantities
    - Copy and paste
  - Mixing of different levels of abstraction
    - Domain layer depending on persistency layer
  - Error management

Lack of knowledge: error (mis)management 4/5

```
try {
    BufferedReader in = new
    BufferedReader(new
        FileReader(“infilename"));
    String str;
    while ((str = in.readLine()) != null) {
        process(str);
    }
    in.close();
 }
 catch (IOException e) { }
```

48

What happens when an IOException is thrown? The handler swallows it, and it is very difficult to spot that an error has happened. Unfortunately, this is a very common pattern found in many production programs.

Lack of knowledge: reinvention of the wheel 5/5

- C++ shared pointers
- String class
- Log frameworks
- Persistency frameworks
- Etc.

49

Reinventing the wheel leads to simplistic solutions in very subtle ways. The “wheel” is not usually part of the domain being modelled, so, when the pressure is on, its development is done cutting corners, and giving under-engineered solutions.

Solutions?

- Continuous learning
  - Books
  - Magazines
  - Professional associations
  - Conferences
- Writing (and reading) software
- Writing articles
- Giving presentations
- Lead by example
- Professional integrity

50

These are just a beginning, and the list is not meant to be exhaustive.

Questions?