Functional specifications

Christophe Delord ~ http://fun.cdsoft.fr

Sunday 28 February 2016

What's wrong today?

The main problems in the everyday life of IT engineers are often the software specifications. The purpose of CDSoft FUN method is quite simple:

Specifications shall be:	but also:
• simple • practical, convenient • usable	• formal • unambiguous • verifiable • safe

Who am I?

I'm a software engineer. I've been working in the IT and aircraft industry for 17 years... writing softwares but not always in a clean and proper way:

- deadlines incompatible with requirements
- silly hypothesis to reduce costs
- and an unefficient philosophy: *do it well at the first time, you'll have no second chance to make it better or to try alternative solutions!*

So I have accumulated some ideas to make it differently.
I also have a passion for free software and sharing good ideas.
More about me: `http://cdsoft.fr`

Why this book?

This book is an opportunity to:

- have time to formalize my ideas
- have fun with things like Arduino, Raspberry Pi, ...
- prove that writing softwares doesn't necessarily take complicated process, methods and tools

So I'll be proud and happy if you decide to help me and fund a little bit this adventure.
I hope I can work part time and have more time for this project.
In return you will get a digital version of the book.

FUN answers

My point of view is that most of the software specifications - even for critical systems - are written:

- in ambiguous natural languages
- in close proprietary inexploitable formats
- with unsafe proprietary office suites

Let's see what's wrong with this and what can be done to improve this situation...

Problem 1: ambiguous natural language

Natural language is nice

- to explain or comment ideas,
- to communicate in every day life. . .

But critical (or not) softwares must be described unambiguously. **Ambiguities** lead to multiple and sometime **inconsistent interpretations** in practice.

Problem 1: ambiguous natural language

- Tools may not be good at understanding natural languages.
- Sometimes people extract information by hand to duplicate it in other documents (design, code, ...). And humans are not good at copying/interpreting/pasting...
- Such projects have a bunch of different and inconsistent documents and the implementation may differ from the original specification.

Problem 2: Close proprietary formats

You can not master a close proprietary format:

- The format may change at every new versions.
- Old versions and tools may not be supported.
- You don't have the source code so old versions will be lost.

Proprietary formats are definitely the worst choice to write documents, especially if documents must be kept for a long time (e.g. 80 years in the aviation industry).

Problem 3: Inexploitable formats

Some formats - proprietary or not - are not suitable, especially office suite documents:

- Very often people don't understand the difference between a text editor and a word processor.
- A word processor may be nice to write a letter to your grand mother.
- But it also saves a lot of insane things in the document to describe the layout which makes it impossible for tools to exploit the document.

Problem 4: Unsafe office suites

- You don't know what some office suites do.
- Do you think that entrusting confidential documents to a country that may spy your activities is a good idea?
- A lot of companies give their documents to some famous proprietary office suites, using computers connected to the internet... No comment.

Proposal 1: Formal language

- Natural language is still used of course but only to comment and explain a formal model!
- A functional language providing essential features:
  - strong and static type systems,
  - type inference,
  - clean mathematical semantics,
  - determinism.
- Such languages are deterministically executable and also unambiguous.
- The specification can be reused for subsequent activities (design, code, test, ...) thanks to a clean unambiguous syntax.
- More tools and less human bugs is not a bad thing.

Proposal 2: Open formats

- Open and simple formats:
  - All documents are written in plain text format (generally UTF-8 text files).
  - This format is so basic that it must be supported for a long long time.
- These formats and the associated tools are generally free (as in freedom) and free (of charge).

Proposal 3: Open formats (again)

- Markdown is a simple document format that any good text editor can read and modify.
- There are tools to convert these formats to HTML, PDF or even some office suite formats but the source document format is, and will always be, exploitable by humans and tools.
- These formats separate clearly the content and the form. Just defines the form once for all and reuse it on every project. And let your engineers focus on the content!

Proposal 4: Safe tools

To write text, you need a text editor:

- Every body has its favorite text editor.
- the FUN method won't force anybody to use a specific editor.
- Just use the one you feel better with and that is safe enough for your activity.
- Every body will then be more productive.

Executable specifications

The advantage to writing **fun**ctional specifications is that they can be executable. It means that they can be executed to:

- debug the model
- test and validate the model
  - the logic of the model can be checked before the real hardware target is ready
- animate a model and show a mock-up to you customer

Some kind of poor man formal method...

Executable test plans

The concept of executable specifications can be applied to tests!
Speaking about *executable* test plans is weird because the rationale of a test is to be executed to check something...
It should not be necessary to explicitly say "*executable* test plans".

Executable test plans

"**Executable** test plan" is not weird. It should be natural to everyone... But people often:

- write a test plan
- implement the tests
- execute the tests
- write a test report
- and make a lot of copy/paste mistakes because of such a long and boring process!

I will show in this book how easy it can be to achieve the same goal with **fun**ctional test plans that are self-executable and that generate test reports themselves.

Roadmap

The next slides describe a provisional roadmap or plan for the book. The main goal is to show how Haskell (and **fun**ctional programming in general) can be used as a multilevel formalization language:

- modeling / specification
- simulation
- coding
- testing

And can be applied to different domains:

- real time embedded softwares
- generic purpose data processors (code generator, . . .)

Roadmap / **Fun**ctional programming

The book will start with a short introduction to functional programming. This introduction will be light because there are already a lot of tutorials and documentation on this topic.
The book will focus on the following points and their advantages:

- pure functional programming: why having no side effect is a good thing?
- strong static type system: how a type system can be used as a sort of formal system to describe a software?

Roadmap / Some applications: time formalization in real time systems (1/2)

Reactive real time systems are often state machines. Their main activity is to modify their current state according to external stimuli. Modelling such systems in a pure functional language (i.e. without any side effect) may seem impossible but we will see how to model time and its consequences on a reactive system.
This exemple will show the advantages of a pure functional language to describe states as well as to reason about their evolution in time.

Roadmap / Some applications: time formalization in real time systems (2/2)

Applications with real time models:

- music generator from data measured by movement sensors
- formalism inspired by SCADE or Matlab Simulink and real time embedded softwares

Roadmap / Some applications: Arduino robot

Another model of reactive system will lead to a concrete realization: an Arduino robot able to sense its environment and react to some stimuli.

- formal specification in Haskell: reactions according to sensors
  - obstacle avoidance
  - moving toward a target
- simulation of the robot in its environment (also simulated)
- actual coding using an Arduino platform

Roadmap / Modeling and Simulation

Modelling of a complete system: classical example of traffic lights and traffic in a road network.

- modelling of a traffic light (state machine, synchronization with its environment, ...)
- modelling of a road-user (behaviour, peak hours, commuting, MTBF of the vehicules, ...)
- modelling of a road (maximal speed, ...)
- global model (road network, users)
- graphic interface to visualize the simulation, interact, inject failures, ...

Roadmap / Some applications: a well specified and tested calculator

This project is a rewriting of a calculator previously written in Lua by CDSoft. The idea is to recode this calculator in Haskell in a cleaner way:

- formal specification in Haskell:
  - strong typing: the type verification is some kind of formal method
  - extreme usage of the compiler to detect more bugs and dead code
- complete unit tests:
  - non regression tests helps for future evolutions
  - code coverage

Roadmap / Some applications: ARINC 665 load generator

The ARINC 665 standard defines a loadable data format. This standard is widely used in aeronautics.
The generation of such loads is highly critical since they contain executables and data for embedded computers on aircrafts.
A **fun**ctional specification can:

- reduce the distance between requirements and implementation (easier traceability)
- be tested in a simpler and safer way

The key idea is to describe requirements along with their properties to let the implementation be testable directly from the formalism of the requirements.

For who?

Whom is this book for?
This method has neither the power of formal methods like Event-B or tools such as Matlab Simulink or SCADE (yet?). Its advantages are:

- Pretty easy to learn
- Power of a strong and static type system
- Expressivity of a functional language
- Based on free tools and languages, so subject to simple and cheap evolutions
- Multiplatform, not forced to any IDE, compatible with any good version control system (git, . . .)

It is therefore aimed at anyone looking for a nice combination of power, efficiency, simplicity and cost.

And now...

Your help is appreciated...

- FUN is still a project.
- I need time to write the book and some examples

I think that crowdfunding is a solution:

- to get more time
- to finish faster

Follow me

- on Twitter to be informed on this project: `https://twitter.com/CDSoftX`
- on my web site: `http://fun.cdsoft.fr`