This article was published (in Russian) in *PC World/RE* magazine. 2003. № 9, pp 52-56; *PC Week/RE* weekly newspaper. 2003. №40, pp 38-39, 42; *Informatics* newspaper for the teachers. 2003. № 44, pp. 22–24, 31; *IT news* newspaper. 2003. № 1 (1), pp.26, 27; *Information control systems journal*. 2003. № 4, pp. 52–56; *Open education journal*. 2003. № 6, pp. 69–76; on the web-sites http://is.ifmo.ru, http://www.softcraft.ru, http://computer–museum.ru, http://zdnet.ru, http://consumer.nm.ru, http://server.md, http://silicontaiga.ru, http://www.opensource-forum.ru/corner.php

New Initiative in Programming Foundation for Open Project Documentation

Anatoly Shalyto

shalyto@mail.ifmo.ru

Saint-Petersburg State University of Information Technologies, Mechanics and Optics,

Computer Technologies Department

http://is.ifmo.ru

The aim of Foundation is to prove the necessity to create the software project documentation. This documentation is not only to contain the description of the desired final software product, but also the circumstantially spread over process of its development. In many cases (at least for the educational purposes) project documentation must be open. To support the Foundation web-site http://is.ifmo.ru was created. In section "Projects" there are a lot of significant examples of documented software projects.

Simplicity needs projection and good taste.

L. Torvalds

It is a mistake to think that programmers wares are programs. Programmers have to produce trustworthy solutions and present it in the form of cogent arguments. Programs source code is just the accompanying material to which these arguments are to be applied to.

E. Dijkstra

In engineering practice, “project” assumes the development of project documentation

Not long ago one of the authors watched how one notable programmer (participant of two *ACM International Collegiate Programming Contest* world finals) has wasted 15 minutes to understand a short program (6 lines of code). He had known that the program was an iterative solution of classical ***Hanoi towers*** problem. After that we found another solution in the Internet, with rather good description of this algorithm.

Open Source Codes and Programs Understandability

In fact open source software does not guarantee program understandability. The Leading analyst of *BASF* Corporation, professor of *Fairleigb Dickinson University (NJ)*, N. Bezrukov, the author of the www.softpanorama.org web-site, believes [1]: “The central question of practical programming is the question of source code understanding.

It is always good to have original code, but the problem is that in most cases it is not enough. Understanding of any non-trivial program requires additional documentation. This need grows exponentially with the size of the source code. Code analysis for recollection of initial project solutions and program understanding are two important branches of programming technology. For instance, try to understand structure of the compiler if you have no definition of formal language, which it compiles.

Everybody, who has participated in large-scale software reengineering projects, remembers the sense of helplessness, which occurs when you see lots of poorly documented (but, maybe, very good written) kilobytes of source code. Availability of the source codes does not help when there is no access to key developers and ideas. If a program is written, for example, in *C* programming language (a low-level one) and its documentation leaves much to be desired then all project solutions dissolve in the coding details. In such situations the value of high-quality documentation like specifications, interfaces definitions and architecture description may raise the value of source code!

Source code is not enough to understand the program. It resulted in appearance of methods which unite code and documentation. One of the most famous attempts of such solutions was taken by D. Knuth in his book “Literate Programming” [2]. Probably, the most well-known prohibited book in the history of computer science was “Commentary on *Unix*: With Source Code” [3], which contains high-level explanation of source codes of *Unix* operating system even with description of used algorithms. This book has been distributed illegally for more then twenty years from the moment of publication in 1977!

If you have not participated in project from its early stages then its complexity and size make it impossible to understand the source code. Understanding of “ancient” code is, probably, one of the most difficult kinds of work that programmers have to do, if there is no documentation or initial developers.”

And one more opinion: “Does any worthy freeware program with readable source codes exist? Although there is a great amount of freeware, there are only a few good pieces of it” [4].

The possible result of this tendency was described by the great Russian mathematician, L. S. Pontriagin: “Only well-done work brings pleasure! If it is done roughly, it causes aversion and bit by bit cultivates in the person immoral attitude to the labour” [5].

Why the Programs Are Not Designed?

So, working without source code is bad, but with source code it is not very good also. There is lack of project documentation, which has to be detailed, fine and accurate. Software source code should be a part of project documentation.

We can remember only three examples, which are produced commonly without design stage: children, works of art and, unfortunately, software.

It is also important that the documentation is used, even when the amount of production is relatively small. Even the single dress is sewed using patterns! Bridges, roads, skyscrapers are built according to documentation, but the software is not.

The situation, which is widely known in software development, can be described by Weinberg’s Second law: “If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization” [6].

Why are there lots of documents, which are issued with devices and hardware? That documentation is easily understandable and designed for specialists with average level of qualification. Such situation allows to redesign the device or hardware (with the documentation) easily even many years after. For programs such sort of documentation is usually missing.

We see the cause of this problem in the following. First of all, devices and hardware are intended for usage by exterior consumers. So the lack of documentation will force developer to spend the rest of his life in the “factory”, but, we hope, he does not plan it. In software development the situation is different. In most cases, the consumer of the source code (not of the final product, but only the code) is the developer (the same organization). The common opinion is that there are no reasons to create good quality documentation inside the organization.

Secondly, devices and hardware are “hard”, and software is “soft”. This means that it is much easier to make changes in the software, than in the hardware, but it does not mean that there is no need for program documentation. In fact the majority of programmers would never read and, especially, write documentation [7].

Experiments show that there are few young programmers, who are able to write program documentation, regardless of the fact that in their universities they have passed examinations on large courses of mathematics. It did not give them ability to expound articulately. For example, the same object can be found in the different parts of documentation as “lamp”, “Lamp”, “bulb” or “Bulb”. The fantasy is unlimited! Development environment helps to avoid such errors in programs, but when writing documentation there is no solution.

Program development has become similar to show-business with its pursuit of profits. Nobody is interested in project future (especially, the far future). The main categories are “profitable–unprofitable” instead of “good–bad”. But in most cases the good technology is also a profitable one.

Unwillingness to write documentation may be caused by the fact that the more closed (undocumented) the project is, the more indispensable the author is!

Such work style, unfortunately, has spread to software development for especially crucial systems.

Programs are written, but not designed! During the design stage, any techniques, which are more complicated than *CRC*-card [8] or usage diagram [9], are considered too intricate and are not used. The programmer can always refuse to use any technology, telling his chief that he could not do it in time” [10].

As a result, even users do not think erroneous program behavior is something extraordinary [11].

At present there exists general opinion in the society, that big houses should be designed and well-documented, but for the software this is not needed. That’s why we can ask a question: why in the projects like “Living House” its components are designed and documented with different level of quality?

In conclusion of this section let us note that present situation didn’t exist at the beginning of the era, when programmers used “big”, ancient machines, first computers. At that time programs were designed and written very thoroughly, because next attempt of its execution may be performed only a day after. So technical progress has resulted in “less responsible” development.

Advantages of Project Documentation

Having extensive project documentation, software developer cannot “control” managers. After programmer’s dismissal, his place can be taken by anyone even with lower level of qualification (and salary), instead of person with higher qualification (as now).

Is it possible to teach how to design and implement programs by books? We think that it is. But now only with separate books: about design [11] and about implementing [12]. Unfortunately, there are almost no books, which cover both stages of software development. Absence of such literature could be filled up by open projects, software with open project documentation. It is similar to new kind of patterns [13]. Project documentation makes refactoring [14] of the software easier.

The project documentation should contain formal specification of program logic, because "things, which have no formal specification, could not be verified and so could not be erroneous" [15]. "If there is no specification then there are no errors" J [16].

Moreover, project documentation is supposed to contain "logs (history of computation), which help to understand program functionality; so theorists of programming are formulating the opinion that set of logs characterize program much better than the source code" [17].

Without project documentation, one of the main advantages of object-oriented programming – code reusing – may result in troubles [18].

And the main point. Project documentation is necessary, because we know from the algorithms theory (Rice's theorem) that in common case it is impossible to prove truth of any non-trivial properties of computing function algorithmically, having only program source code. "Non-trivial property" means that there are programs, which possess this property and which do not.

According to Turing, program understanding (opposite to its execution) requires "astuteness and inventiveness". But source code analysis cannot be automated and human analysis requires huge amount of time. So it is impossible to make any conclusions about program properties!

Mathematicians found solution of similar problem in the antiquity. We mean writing proofs with the help of human language (documenting proofs). This method differs Greek mathematical school from the Egyptian school. In the Egyptian school, solutions of, for example, geometrical tasks were presented as a figure with "explanatory" legend: "Look!" It is similar to understand the program only with the help of its source code.

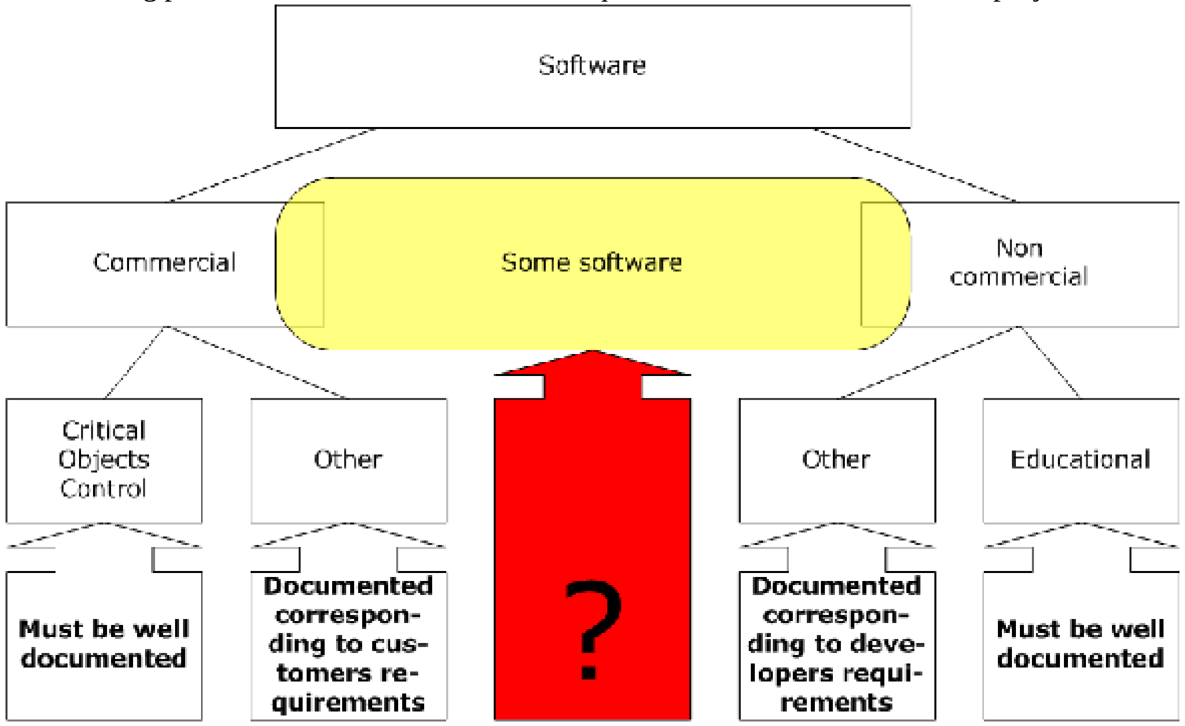
To allow others to understand the program logic, it is necessary to give a detailed description for the person with average level of qualification. This description has to cover program development and its static and dynamic properties. It actually represents project documentation and proves that the program computes necessary function (corresponds to requirements specification).

We should note that "project documentation" does not mean "operating documentation" (user's manual, software development kit or something of this kind)

The preceding facts make it obvious that software project documentation is indispensable. For different projects it can be secret, partly open or open. But for educational purposes project documentation should be open.

Variety of requirements for the project documentation

The following picture shows the documentation requirements for different kinds of projects.



Requirements for the project documentation

Let’s comment on the picture. Documentation for critical objects control software can be divided in two parts – basic, which should be created according to the standards and extra, which is provided to the customer. Other commercial projects are usually documented at customer’s request.

The first type of commercial projects must be well-documented, but it is not always so, because there is not much experience in writing such kind of documentation and people are not used to do it, unlike in IBM and some other companies.

Lack of time and funds are often the reasons for poorly documented projects. The customer usually requests only some manuals and operating guides.

Non commercial software, which is not educational, is documented at the will of the developer. In most of the cases, the developer doesn’t see the need to document the project, except for writing comments in the source code and making understandable names for functions and variables.

So we face joyless facts, especially in the cases, when non commercial software is used together with the commercial software without documentation.

In the described approach, the code should be based on the documentation, but not vice-versa. Meanwhile the documentation should describe not only the product itself, but also the design process.

Foundation for Open Project Documentation

One of the authors (Anatoly Shalyto) declared "Foundation for Open Project Documentation" at the opening of North-Eastern European semifinal competitions of *ACM International Collegiate Programming Contest* (Saint-Petersburg, November 2002). For support and propagation of the foundation http://is.ifmo.ru web-site was created.

At the *Computer Technologies Department* of *Saint-Petersburg State University of Information Technologies, Mechanics and Optics* the special pedagogical experiment began. Students were divided into nearly 40 groups (one or two persons in each group). Each group was to develop some project, using automata-oriented programming technology (programming with explicit state selection) [19]. Created systems with complete project documentation are published on http://is.ifmo.ru web-site ("Projects" section). Many of them are already available.

Let us list some of finished projects or projects in development:

- concept of algorithm visualization for teaching discrete mathematics and programming;
- implementation of interactive scripts for educational animation using *Macromedia Flash*;
- environment for teaching and testing students in arbitrary subjects (with example for English language lessons);
- combined usage of compiler development theory and automata programming;
- skeleton animation;
- usage of *XML* for describing appearance of video player (project home site: http://www.crystalplayer.com);
- control systems for different devices (diesel generator, elevator, turnstile, coded lock, cash dispenser, traffic lights, coffee-machine, phone and many others);
- bank security system;
- client-server architecture, using automata programming;
- graphical user interfaces, using automata programming;
- SMTP-protocol implementation;
- classical parallel problems: "Synchronization of the Chain of Shooters" and "Task about Philosophers Dinner";
- games: "Robocode" [20], "Terrarium", "CodeRally", "Sea Wars", "Lines", "Automatic Bomber", "Slotmachine" and "Bank" ("Zavalinka")).

Important note: among hundreds of tanks for the “Robocode” game, only our tank has full project documentation [21]. The same situation is with the “Terrarium” [22] game.

We suppose that if the world is going to open source codes then there will be time for wide usage of open project documentation. This will allow to read project documents instead of dealing with source code.

The Tale at the End

During the development of one of our projects (“Sea Wars” game), after eighth session of correction (and that is not a record) project author, student of *Computer Technologies Department*, lost his smile like a Cheshire cat. We were satisfied by the project quality. Program has good user interface, works well and is accurately documented.

But open project documentation shows both advantages and disadvantages of the program. One notable programmer (prize-winner of competitions of *ACM International Collegiate Programming Contest*) looked through source code and project author began to redesign it again. This process (with breaks) continues more then half a year and we hope that in this project everything will be perfect like in Chekhov’s stories: the face (user interface), the clothes (documentation), the soul (source code) and thoughts (program work).

Why the documentation should be opened?

The “open” project documentation should exist and should be available for futher usage and possibly for futher modification.

“Foundation for open project documentation” is free, anyone can support it. But it differs from “Free Software Foundation” and “Open Source Foundation”, because the ideas and concepts of open project documentation can be used not only in free software, but in commercial, secret and other projects.

Conclusion

The great amount of work required to write extensive documentation makes it hard for the open project documentation to be used in software development show-business. Offered technology is “hard”, but only “easy” and “agile” methods [23] become popular now.

Nevertheless there are some areas of programming, where it is impossible to work without “hard” technologies. So new people, who need and like software with good project documentation, appear.

One of the students, after he had seen project documentation, written according to our approach, for the first time, said that it was more detailed than TV-set’s documentation. He supposed that submarines were documented in same manner.

Even if usage of project documentation does not become popular, it is very significant from the pedagogical point of view. Development of well documented projects is useful for the participants. It is also important for those, who just look through projects because of their cognitive and aesthetic value. You know that not all visitors of the museum are artists.

Let us finish with the quotation of one opinion about our approach: “There was a lack of Open Project Documentation. Almost all documentation for the commercial projects, unfortunately, serves as customer’s property and they do not hurry to proclaim it. That is the reason for a small amount of real projects in the Internet. There are lots of web-sites with source code, but next to nothing with the project solutions. I looked into the “Projects” section on your site (http://is.ifmo.ru) and compared my solutions with the offered. It is a pity that I had no access to these works earlier. I would not spend so much time for design and development!” [24]

Finally, we would like to point out that the article reflects the tendency of the last years: the software development seems to start “growing up” — transforming from the art through the science [25, 26] to the engineering craft [27, 28].

This research is supported by Russian Fund for Fundamental Investigations (grant №02-07-90114, “Technology of Automata Programming Development”).

The materials of this paper will be reported on the Linux Summit 2004 (http://www.linuxsummit.org/summit2004_program.shtml).

References

1. *Bezrukov N.* Reiterated Look at “Council” and “Market” // BYTE/Russia. 2000. № 8.
2. *Knuth D.* Literate Programming. Stanford: Center for the Study of Language and Information, 1992.
3. *Lions J.* Commentary on UNIX: With Source Code. Annabooks, 1977.
4. *Protasov P.* From Below // Computerra. 2003. № 19.
5. Researcher of “Steering Wheel” // Informatics. 2003. № 11.
6. *Bloch A.* Murphy's Law // ECO. 1983. № 1-3.
7. *Demin V.* Problems of Working of Russian Developers on the West // PC Week/Russian Edition. 2001. № 32.
8. *Badd T.* Object-oriented Programming. SPb.: Piter, 1997.
9. *Booch T., Rambo D., Jacobson A.* UML. Users Manual. M.: DMK, 2000.
10. *Fowler M.* New Methods of Programming // http://www.spin.org.ua.
11. *Booch G.* Future Creation // Open Systems (Otkrytye sistemy). 2001. № 12.
12. *Stroustrup B.* The C++ Programming Language. M.: Binomial (Binom), Addison-Wesley, 2000.

13. *Gamma E., Helm R., Johnson R., Vlissidis J.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
14. *Fowler M., Beck K., Brant J., Opdyke W., Roberts D.* Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.
15. *Zaitsev S.* Description and Implementation of Computer Nets Protocols. M.: Science (Nauka), 1989.
16. *Allen E.* Bug Patterns in Java. APress, 2002.
17. *Ershov A.* Mixed Computations // In the World of Science (V mire nauki). 1984. № 6.
18. *Telles M.A., Telles M., Hsieh U.* The Science of Debugging. The Coriolis Group, 2001.
19. *Shalyto A., Tukkel N.* Programming with explicit state separation // World of PC (Mir PK). 2001, vol. 8, 9. http://is.ifmo.ru (section "Articles").
20. *Shalyto A., Tukkel N.* Tanks and Automata // BYTE/Russia. 2003. № 2. http://is.ifmo.ru (section "Articles").
21. *Ozerov A.* Four Tankmen and the Computer // Magic of PC (Magia PK). 2002. № 11. http://is.ifmo.ru (section "About Us").
22. *Markov S., Shalyto A.* Controlling System for Herbivorous Animal for "Terrarium" Game. http://is.ifmo.ru (section "Project").
23. *Cockburn A.* Agile Software Development. NJ: Addison-Wesley, 2001.
24. *Trofimov S.* E-mail: info@caseclub.ru.
25. *Knuth D.* The Art of Computer Programming. MA: Addison-Wesley, 1998.
26. *Kazakov M., Korneev G., Shalyto A.* Using Finite Automata in Developing Logic of Algorithm Visualizers // Telecommunication and Informatization of Education. 2003, vol 6., http://is.ifmo.ru (section "Articles")
27. *Sommervill I.* Software Engineering MA: Addison-Wesley, 2001.
28. *Braude E.J.* Software Engineering: An Object Oriented Perspective. NY: John Wiley&Sons, 2001.