

# **Open Source Software Engineering**

## **II OSWC**

Alexander Dymo,

Information Control Systems and Technologies Department,  
NUS (National University of Shipbuilding), Mykolayiv, Ukraine

<http://www.ki-inform.com/~adymo>

### **Introduction**


Open source software today became widespread among not only volunteer developers but also among commercial and governmental organizations. Whereas open source communities developed certain engineering practices, no serious effort was undertaken to research them. Thus commercial and governmental organizations adopting open source development model meet noticeable difficulties in open source project management. Basically only disorderly data is available about OSS life cycle and team management and little is known on how to estimate cost and effort for such projects. The goal of this paper is to provide concise information about open source as a software development method. It will describe the method in general, give the sketch of the life cycle model, provide information about team management and finally present the model for cost estimation.

### **Open Source as an Agile Method**

In general, open source can be considered as another Agile software development method. There are several point that prove the statement above. Open source world values are the same values stated in the Agile Manifesto [1]:

- individuals and interactions over processes and tools:  
OSS was started as a pure-volunteer effort and eventually adopted the “meritocracy” thus being pure individual-oriented;
- working software over comprehensive documentation:  
“getting the work done” was and is the main governing principle for all OSS developers;
- customer collaboration over contract negotiation:  
the best case for OSS “customer is a developer” is the ideal example of customer collaboration, also successful OSS projects always show a great deal of user (read “customer”) collaboration;
- responding to change over following a plan:  
last-minute changes before the freeze are way too common for OSS.

Also the most important Agile concepts as highlighted by Boehm and Turner in [2] are widely

supported in OSS development practice. Change was always the source for inspiration for an OSS developer (*embracing change*). The study on distribution  developers in usual OSS project shows that most people work rather on new features than on fixing bugs. They also tend to work on new versions of the software than to continue improving old ones. This tendency implies *fast development cycle* because new features are usually implemented in newer versions. OSS teams rather deliver more versions of a software than make the user wait. *YAGNI* (“You Aren't Going to Need It”) principle has been employed by OSS makers for ages with all its pros and cons. And finally, a specific “culture” is being observed among OSS world (with its own leaders like Richard Stallman, Eric Raymond, Linus Torvalds). Most teams have their own subculture which is shared by the majority of developers. Such cultural groups concentrate knowledge, vision and moral of a project (*tacit knowledge*).

Points stated above show that OSS development method can and should be considered as another Agile development method available for project managers. The rest of this paper gives answers to common questions about software development method:

- how is the process defined (i.e. what is the life cycle model)?
- how to gather a successful team of developers?
- how to estimate project effort and cost?

## Life Cycle Model

The definition of the software development process is best given using the life cycle model which in case of OSS looks very similar to the spiral model presented by Boehm [3]. The OSS model is illustrated in Figure 1.

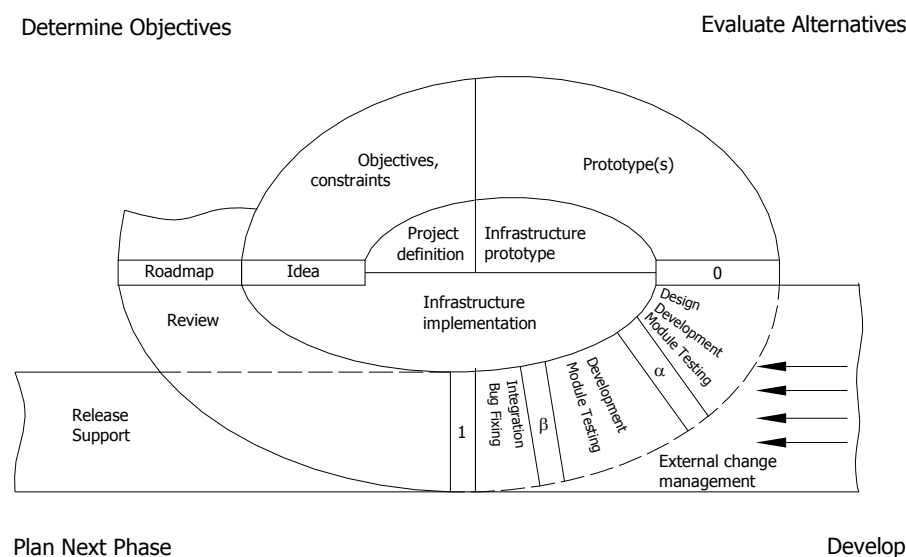


Figure 1. Open Source Model of the Software Process

A spiral begins with the **Project Definition** phase where the key idea is stated (“we’re going to build the best IDE” would be a good example of such an idea). During the next two phases the project **infrastructure** should be designed, **prototyped and implemented**. The infrastructure (policies, websites, mailing lists, source code repositories) is a vital part of an OSS project engineering because the better infrastructure is the more efficient development team becomes. After infrastructure is implemented, **objectives and constraints** considered, the first **prototype** is produced. Usually it is released as a **version 0** (0.1 or 0.0.1) to the public. The goal of this early release is to attract more developers and intrigue potential users. After that development is started, the **code** is being produced, **tested**, external code patches applied and more developers **involved from the outside**. At certain time the team leader (project maintainer or release coordinator) decides on “**alpha**” release. The goal of alpha release is usually to stabilize the code base and stop the inflow of features. After the “freeze” is over, the development is headed to the “**beta**” release. Beta release is usually more stable because it is aimed for users. All feature changes are forbidden from the “beta” release point. Only bug fixes are allowed. After all fixing and testing is done the **first release** is out to the public. This release enters its own maintenance cycle (that looks like similar to the cycle on Figure 1). The development continues through the **review** phase to the next iteration of a spiral heading to the version 2 of the software.

## Open Source Team Management

Open source teams are known to be successful – they produce more value using less resources than many analogous commercial organizations. Also practice gives evidence that lots of almost “dead” OSS projects “magically” survived. The answer to the question why it happens is more pragmatic than “magic”. OSS teams appear to be self-organized teams. Self-organization is the key to understanding reasons why OSS teams are so effective (and so vulnerable under certain circumstances). Detailed research gives hints on how to manage open source teams so they become even more successful.

Self-organization here is considered in Ashby's sense [4]. Ashby was the first who defined the mechanism of self-organization as shown on Figure 2.

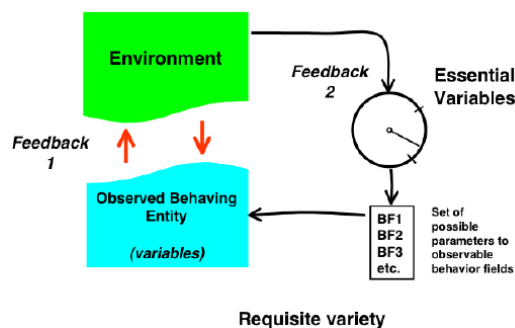






Figure 2. Design for self-organized system

This diagram depicts minimum conditions that exist in an adaptable system with increased chances of survival. Ashby in [4] concluded that such systems must have at least two feedback loops. The first feedback loop provides means of immediate communication between the system (entity) and its environment. For software teams example for such a loop would be iterative communication between the user (customer) and developers. Second feedback loop starts from the environment and encompasses sources of important but rare changes that could make the entity (team) to die (fail). Those changes are shown as “essential variables”. Self-organized system should react when essential variables are driven outside their normal limits by changing one of their parameters (variables not included in the system and usually considered constant). This enables survival.

Geoghegan and Pangaro shown in [5] that any social system (including software teams) could be made self-organized. The research into OSS teams has shown that they developed the second feedback loop with an environment. OSS team essential variables and possible reactions are summarized in the table below. For each variables the bold line indicates the range of “normal” values.

<i>Essential variable</i>	
<i>Parameter change for low values</i>	<i>Parameter change for high values</i>
	
more enterprising developers	more pragmatic developers
	
new roadmap with detailed steps to get feasible results	pragmatic roadmap, feature prototyping, proving
	
review of successful use-cases	review of unsuccessful use-cases, demand for discussion, prototyping
	

<i>Essential variable</i>	
<i>Parameter change for low values</i>	<i>Parameter change for high values</i>
more commits, even “artificial”	demand for API docs, feature overviews
<div> <div></div> <div>disorganized</div> <div>organized</div> <div>cool</div> <div>prominent individual driven</div> </div>	
new team leader election by meritocracy rules	more emphasis on team spirit, importance of all members
<div> <div></div> <div>don't show</div> <div>usable product</div> <div>cool product</div> <div>super product</div> </div>	
more users, reviewers, conference participation	more criticism (make sure criticism is well-perceived among team members)

A polar graph representing the stability area of OSS projects is shown on Figure 3.

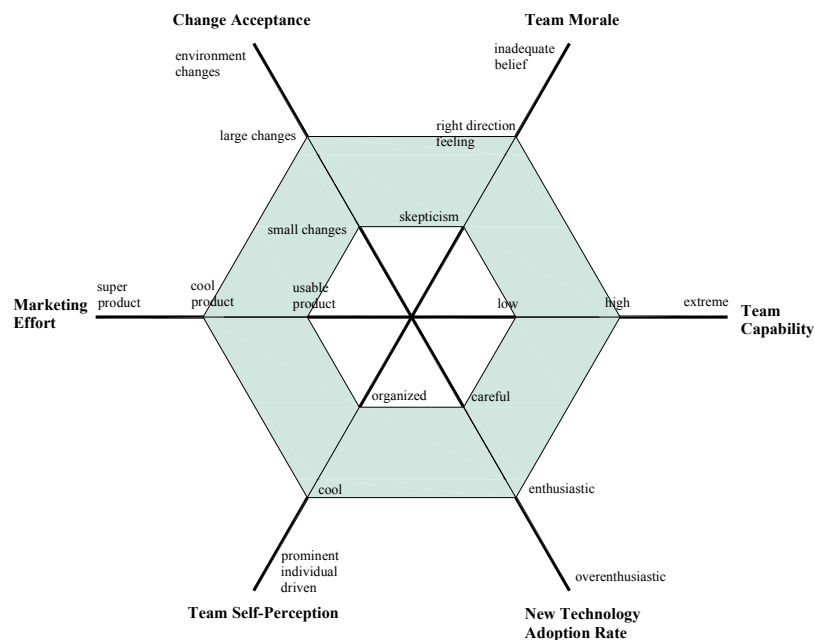


Figure 3. Essential OSS project variables and stability area polar graph

### OSS Development Cost Estimation

As described above, open source projects tend to spend less results and produce more value than commercial, closed-source projects. This positive fact however imposes certain OSS project management difficulties. It is no longer possible to use even detailed COCOMO models [6] for cost estimation because they do not give valid results. Troubles of COCOMO model appliance are shown

on Figure 4. While the charts on Figure 4 show only two projects, the overall tendency is kept among the rest of them.

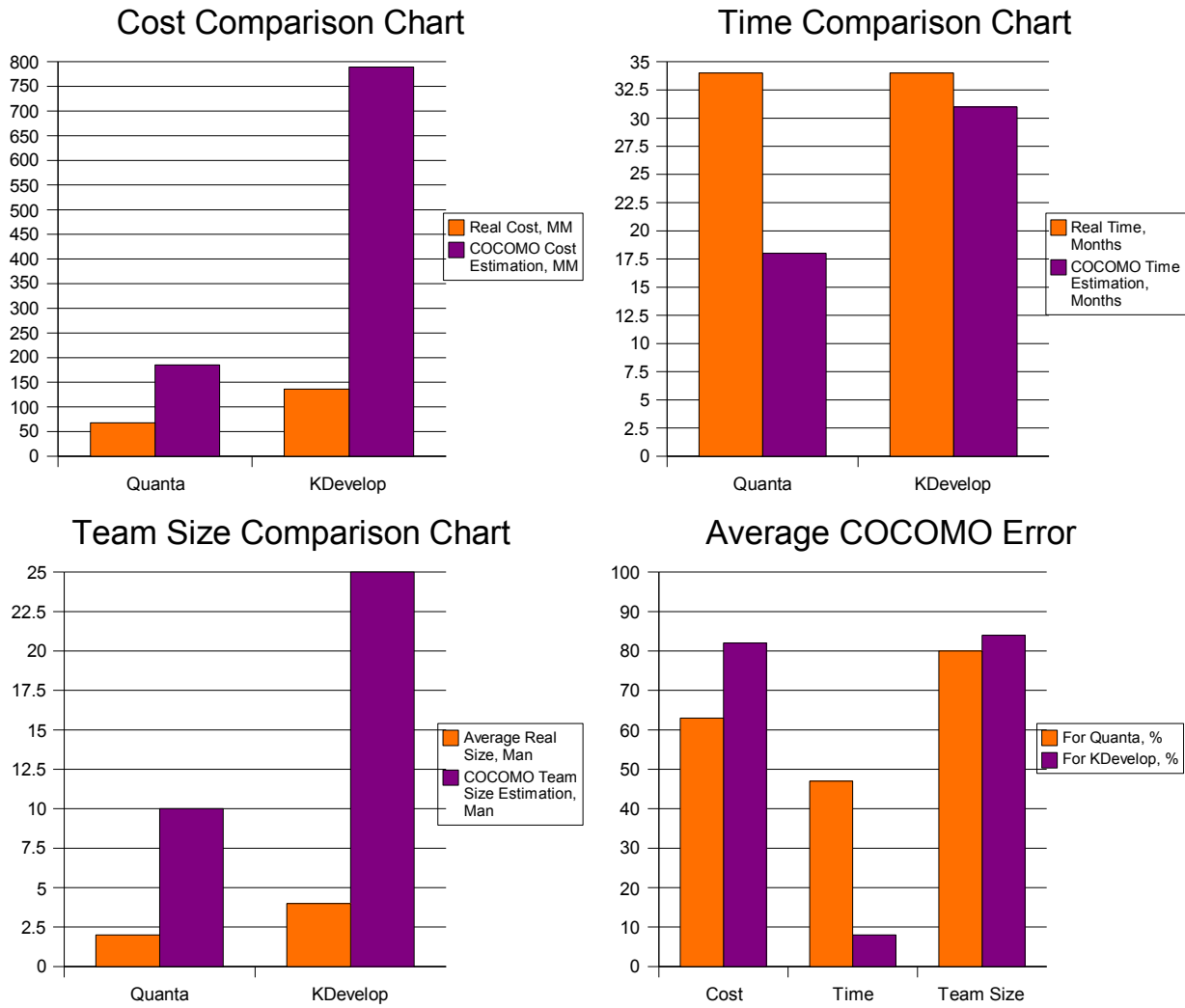


Figure 4. Validity of COCOMO cost estimations for two OSS projects: Quanta and KDevelop

Figure 4 obviously tells that a new model for OSS cost estimation should be built. The basis for building OSS cost models was first presented in [7]. The paper describes how analog models can be built and used for software engineering in general and OSS engineering in particular.

Analog model is presented in form of equation:

$$Dts = A \cdot Dlcom^e \cdot Dprod^f \cdot Dreus^j \cdot Dqua^i,$$

**Dts** – criterion of time and size:

$$Dts = \frac{t \cdot lf}{sz}; [Dts] = \frac{day}{fp} \cdot \frac{fp}{day} = 1.$$

**Dlcom** – criterion of complexity:

$$Dlcom = \frac{dcx \cdot dc}{lf}; [Dlcom] = \frac{fp}{man \cdot day} \cdot man \cdot \frac{day}{fp} = 1;$$

**Dprod** – criterion of productivity:

$$Dprod = \frac{dp}{dcx}; [Dprod] = \frac{fp}{man \cdot day} \cdot \frac{man \cdot day}{fp} = 1;$$

**Dreus** – criterion of reusability:

$$Dreus = \frac{cc}{rc \cdot rv}; [Dreus] = comp \cdot \frac{1}{func \cdot comp} = 1;$$

**Dqua** – criterion of quality:

$$Dqua = \frac{dpx}{q \cdot dcx}; [Dqua] = \frac{bug}{man \cdot day} \cdot \frac{fp}{bug} \cdot \frac{man \cdot day}{fp} = 1.$$

Other values that form criterion equations are:

**t** – time of development,  $[t] = \text{day}$ ;

**sz** – software size,  $[sz] = \text{fp (function points)}$ ;

**dcx** – complexity by Shaffer [8] – a number of function points that a developer should produce per day to meet the deadline,  $[dcx] = \text{fp} / (\text{man} \cdot \text{day})$ ;

**dpx** – developer experience by Curtice [8] – a number of bugs a developer can fix during the day on a test case,  $[dpx] = \text{bug} / (\text{man} \cdot \text{day})$ ;

**lf** – language factor – an average number of function points can be produced per day,  $[lf] = \text{fp} / \text{day}$ ;

**dp** – developer productivity by Albrecht – a number of function points a developer produces per day,  $[dp] = \text{fp} / (\text{man} \cdot \text{day})$ ;

**dc** – number of developers,  $[dc] = \text{man}$ ;

**cc** – component count,  $[cc] = \text{comp(onents)}$ ;

**q** – quality by Shafer [8] – a number of bugs estimated per function point,  $[q] = \text{bug} / \text{fp}$ ;

**rc** – requirement complexity - a number of necessary, steady functional requirements,  $[rc] = \text{func}$ ;

**rv** - reuse volume - the specific cover of functional requirements by ready-to-use components,  $[rv] = \text{comp} / \text{func}$ .

Such analog model can be tailored for specific kinds of software projects (like IDE, text editors, etc.). To tailor a model is to define the exponents in the model equation. Currently the exponents for major types of OSS projects are being investigated at NUS. Once the research is finished, the ready to use model equations will be available to use by open source managers and team leaders.

## Conclusions

OSS is Not a Silver Bullet. Open source did not kill the werewolf of software development and it is not going to. Experimental data indicates considerable (up to 80%) decrease of development cost and team size for OSS projects with a noticeable increase in development time (up to 40%). Analysis shows that open source is another agile development method with incremental spiral life cycle. The decrease of team size necessary to complete the project is achieved through usual agile methods and most importantly through self-organization mechanisms. Proper understanding of the nature of OSS development method presented in this paper will enable managers to select it when appropriate and apply with care and knowledge. New analog cost models will help them to estimate costs (time, effort, schedule) of OSS production and complete even more successful projects.

## References

- [1] *Manifesto for Agile Software Development*.
- [2] Barry Boehm, Richard Turner. 2003. *Balancing Agility and Discipline. A Guide for the Perplexed*. Boston: Addison-Wesley.
- [3] Barry Boehm. *A Spiral Model of Software Development and Enhancement*. ACM SIGSOFT Engineering Notes vol.11 no 4. Aug 1986. pp. 14-24.
- [4] Ashby, W. Ross. 1952. *Design for a Brain*. Chapman and Hall. London.
- [5] Michael C Geoghegan, Paul Pangaro. *Design for a Self-Regenerating Organization*. Ashby Centenary Conference. March 4-6, 2004, University of Illinois, Urbana.
- [6] Barry Boehm. 1981. *Software Engineering Economics*. Upper Saddle River, New Jersey: Prentice Hall PTR.
- [7] Alexander Dymo. *Application of Similarity Theory to Software Engineering*. Scientific papers #6. February 2006, National University of Shipbuilding, Ukraine (final version is available only in Russian language, preliminary English translation: [http://www.ki-inform.com/~adymo/similarity\\_en.pdf](http://www.ki-inform.com/~adymo/similarity_en.pdf)).
- [8] Robert Futrell, Donald Shafer, Linda Shafer. 2002. *Quality Software Project Management*. Upper Saddle River, New Jersey: Prentice Hall PTR.