ПРИЛОЖЕНИЕ № 5

Таблица № 1. Материал за развитие на знания и умения по темата „Стейкхолдър управление при проекти с отворен код“ *(английски език)*

Factors	Contents	*Context*
Organizational		
Value-adding stakeholder relationships	**Understanding of the personal drive for participation in the project**	*Educaton plays a key role, as well – a lot of people join a project just to see how something works (part of code, a process, etc.).*
	Understanding of the existing (and potential) open source business models	*Many factors can influence the interest of the community towards the project. For example, projects that target developers (e.g., development tools), are found to be more successful, compared to application and/or system software (Rehman, 2006). However, those are project characeristics that cannot be influenced – it should not to advised that a project is dropped due to its filed of application or type of product in mind. It is possible to turn a niche project into a success, even against the odds, as long as there is a sustainable business model around it.*
	Two-way communication within the broader user community	*Although there are many channels by means of which the broader user community can influence the course of development (Scacchi, 2002), stimulating more interaction among user and developer communities would be quite useful. Not only that engaging into meaningful communication outside the active membership base results in higher product popularity (which on its end positively impacts user contribution - Colazo, 2007), but it also creates broader set of strategic opportunities, based on the feedback acquired.*
	Promoting the mechanisms and benefits of the open source model	*This includes 'evangelism' throughout companies, as well, where executive and line management buy-in, awareness and support is a must. To embrace the open source model, each company needs to ensure that it adopts appropriate OSS governance (internal rules and regulations) aligned to organisation's strategy, which may be an obstacle, given the lack of awareness on open source governance in general.*
Legal Strategy	**Implementing permissive project license**	*Studies show that license restrictiveness is negatively associated with user interest (Subramaniam et al., 2009).*
	Registering a trademark	*Only by protecting the brand, truly efficient branding efforts can be achieved (the various methods of promoting the product, and the brand). The logo, style guid, graphics, official colors or fonts, they would all be a valuable project/community asset in the long run.*
	Attracting and managing financial donations	*While time is the key scarce resource in consideration, when it comes to open source projects (with incentives for donating time, related to career concerns, ego gratification, and others), an increasing number of open*

		source projects are considering monetary donations, which if pursued, need to be properly managed - a problem on its own. Donations are proven to impact directly the performance in user, as well as in developer-oriented projects, e.g. developers may feel grateful for donators' appreciation and hence become more actively involved in the projects (Hahn & Zhang, 2005).
Awareness of the broader project	**Conducting preliminary research of existing open source projects**	*What can be done, is to conduct a preliminary research of existing open source projects in the filed, that may or may not justify the need to launch a new one (instead of joining existing). It will also provide important information for benchmarking purposes, enabling more precise mapping of the involved stakeholders and any gaps identified.*
	Coalition-building as a foundation for a sustainable team	*Poor coordination is often described as one of the main obstacles on the open source projects' way to success, but effective coordination starts with a solid foundation, that is coalition-building.*
	Awareness of the marketing aspects of the project	*Koch et al. (2003) claim that to establish a strong network with its after-effects the number of participants (developers, contributors and the user base) in an open source project, you must go beyond a critical threshold, i.e. there must be active marketing and promotion of the project. However, marketing is more than that. Being „a set of institutions, and processes for creating, communicating, delivering, and exchanging offerings of value" (AMA, 2007), whatever the marketing orientation of the project is, it brings with itself a great number of stakeholders to take into consideration.*
	Considering financial sponsorship needs and opportunities	*Studies show that having a (financial) sponsor is positively associated with user interest (Stewart et al., 2006). In addition to that, project teams are looking more and more into alternative/ad-hoc funding, which needs proper governance (grants, crowdfunding, etc.), allowing monetary incentives to be offered to developers to encourage their participation and overcome growth-related challenges.*
Coherent project setup	**Clarity on governance structure and decision-making model in place**	*In OS management, organization may take extremely distant forms - one may encounter "monarchies", "oligarchies", and "democracies", with community members voting to decide upon project evolution (Stamelos, 2014). Other projects may inherit unique features from their sponsors (an example of such a structure is the concept of project incubation, practiced by the Apache Software Foundation and Apereo Foundation).*
	Applying proven project management practices	*It may come as a surprise to part of the open source community, but leveraging common management practices in particular project areas (such as human resource management), and in certain development fields (e.g., release management, bug fixing) is proven to have impact on open source project performance (Hahn & Zhang, 2005; Ghapanchi and Aurum, 2010).*
	Strategizing and commitment	*Michael Ferris, Director of Business Architecture at RedHat, suggests that any individual or business, considering the open source model, should ask themselves what do they want out of this engagement (e.g. to build a community of users, build a community of contributors, seed the market for a business, build a business*

		on the code itself, etc.) – a strategic question, impacting the value that the sponsor provide, based on the open source decisions that are made, as well as impacting the relationships throughout the stakeholder network.
	Network strategy	*This topic is reffered to as 'social network ties' and 'quality of social ties' – the number of direct and indirect ties (Singh et al., 2007; Hahn et al., 2008), as well as the extent to which a OSS project is connected with other important OSS projects (Grewal et al., 2006).*
Comprehensive tackling of intergroup and intragroup conflicts	**Charter-based governance of the project**	*Defining project's preferences and priorities helps avoid potential issues later on. A project charter will boost the sense of community, paving the way to a clear dispute resolution, given that the common goals are set and democratic decision making mechanisms can be formed around them.*
	Addressing disputes and conflicts within the community, based on a code of conduct	*When conflicts prevail, there are cases when a developer will secede from the original project to form a new development initiative (or fork) based on the source code (Raymond, 2001). Only by addressing any disputes and conflicts, based on a solid code of conduct, one might expect that the worst-case scenario would get avoided. A dissatisfied part of the community leaving the project may not be seen as an irrevarsable downfall, but a code of conduct might prevent the wrong people engaging with the initiative to start with.*
	Managing Growth	*The more the contributions from the community – the bigger the chances for project success, as Long proved through empirical research (2004). However, managing growth becomes a key necessity for fast-growing initiatives without which expansion in terms of project size and complexity becomes a factor of risk.*
Openness in project governance / operations	**Comprehensive project documentation**	*Comprehensive means both in terms of contributions, and contributors, e.g. user contributed documentation. Many teams use wikis for how-tos, worksh-for-me platform testing results and other collaborative use-cases, but there are more advanced solutions (which include wiki) that can be implemented to enable collaboration on core project documentation and beyond (files, ideas, minutes, specs, mockups, diagrams, etc.). Some of those solutions include "Redmine" on the open source end, and Atlassian's "Confluence" on the propriatory side.*
	Active communication within the project community (multi-channel)	*Long (2006) found that the number of messages in the forums is positively associated with OSS project success, similar to the number of bug and patch reports, coming from non-core developers. The point here is about enabling the natural flow of communication among members, but also about facilitating and encouraging the discussion on both strategic and operational level throughout different channels.*
	Transparency in the business practices	*The ways to build value „beyond the bits", whether through access to updates, service, support, certification, ecosystem relationships, and others, determine how far the sponsor is willing to embrace the open source community (Ferris, 2013). No matter the choice, it is critical to apply those business practices in a transparent manner that do not pose a threat to the levels of trust within the community, as well as to your perceived integrity, as communities tend to vote (confidence) with their feet.*

	"Gateway" project website	*While the project repository might be th best fit for contributors, stakeholders, which are not tech saavy, may be excluded if an up-to-date "gateway" project website is not present to provide all the necessary links to development, support, communications and other resources. Martin (2015) puts the web site on top of OSS project infrastructure essential components' list, along with source code repository system, mailing lists, IRC/instant messaging, bug and feature tracking, automated build and test system, etc.*
Community building agenda	**Nurturing community identity (common values and sense of belonging)**	*Many researchers have focused on the benefits of network ties in the case of open source software development. Co-membership among project teams is proven to be an effective mechanism for building network ties, through which knowledge and expertise glows across projects (Peng et al., 2013), but nurturing intra-community identity is equally important. Without strong membership bonds, based on common values and a sense of belonging, communities may simply dissolve – the more connected they are to other projects, the easier.*
	Organizing and conducting project events (online and offline)	*Subject-specific face-to-face conferences, developer meetings and workshops are among the most common value-adding activities to grow and sustain a successful project community.*
	Providing mentoring and support	*Support for new participants by experienced community members is mostly about 'kick-starting' a self-organization process within the community (the spontaneous appear of order/global coordination out of local level interactions), extrapolating on the basis of feedback loops and network effects.*
	Providing opportunities for personal and career development	*As an example, the core team can encourage the community to use the publicly available project-specific communication or dissemination mechanisms. This would not only limit the chance to lose major decision making processes and information dissemination in private conversations ( ), but would also . A dedicated place on the project site, or on the online forum might be a small, but effective step in the right direction.*
Coherent contributor engagement strategy	**Recruiting newcomers to ensure long-term sustainability of the project community**	*The same concerns the core team of developers - leader succession may become an issue as the original developer may have fulfilled his/her original requirements (technical or personal) and may lose interest. In such cases, an orderly succession can occur only if there are other members, interested in assuming the leadership role (Wynn, 2004).*
	Precise mission statement, resonating with the community members	*The strategic direction of an open source project, as defined in a precise mission statement, may be what best keeps the commitment going, especially given the blurred timeline and ambiguous stakeholders' goals that can lead to the disintegration of the community as a whole.*
	Problem-oriented basis of the project	*As far as commitment is concerned, we can agree with Schweick and English (2007;2013) that the minimum requirement in order to succeed in OS is to be active. Success is variable in nature - it can continue, even grow, but also fade and die (Mattila, Mehtonen, 2014). Hence, we need a problem-oriented basis of the project, which ensures that as long as this problem is of importance to the community, it will continue to devote time and resources on initiaitves that are intended to resolve it.*

	Assigning clear responsibilities	*Assigning clear responsibilities directly addresses another success factor, derived from research – "service quality", i.e. contributors/developers capability to promptly provide information, services, and solutions to reported problems (Lee et al., 2009; Stewart and Gosain, 2006).*
Technical		
Technical Cohesiveness	**Equilibrium between the level of the project and community member's expectations**	*Contributor's experience (e.g., how experienced a project developer is in terms of the skills or even the time span of writing programs) is a positive force, driving greater efficiency and activity (Hahn & Zhang, 2005), but it also suggests certain prefferences/expectations on behalf of the developers, regarding key project characterisitcs, such as project's goals and quality of work. Significant mismatch between those (due to expectations not being managed, and eventually met) may result in dissatisfaction and cause high overall instability (due to the increased contributor turnover).*
	Enabling project scalability through modular organization of the project	*Modularity can be defined in terms of technical modularity (coupling of modules) and org. modularity (coupling of module owners). Research show that product decomposition does not only enable scalability, but is also directly related to project performance (Liu, 2008).*
	Taking account of the technological limitations	*Strategy planning should take into consideration technologically-imposed constraints in the scope and implementation of the project (e.g., projects using less common programming languages are found to be less active, in less advanced state of development, less used and generally less successful - Crowston & Scozzi, 2002).*
	Ecosystem participation	*Production wise, some projects simply cannot reach their full potential without a stronger engagement from technology suppliers, tools producers, and perifferal stakeholders. In fact, in an open hardware community survey (Element 14 Community, 2015), this is brought up as the most critical challenge to expanding the open source hardware movement.*
Participation-oriented tech. governance	**Frequent version releases**	*Frequent version releases are considered as a sign of project vitality, which impacts popularity and shapes the behavior of the community – it rewards and triggers acitiveness. Researchers claim that the more active a project is in terms of new releases and making announcements, the more attention it receives from the community (Stewart and Ammeter, 2002).*
	Managing complexity	*Whenever complexity is/becomes an inevitable feature of the project, there are a number of code/design characteristics to pursue in order to manage it successfully (e.g., understandability, completeness, conciseness, consistency, maintainability, testability, usability, reliability, structuredness, etc. – Crowston et al., 2003).*
	Accuratelly describing/filing bugs	*In the context of open source software, testing by as many users as possible is an important mechanism to tackle potential issues, but it does not help when it comes to making people contribute when they simply cannot understand the project's challenges well enough. Therefore, accurately describing / filling bugs and issues to*

		work on is regarded as a highly important characteristic of project success in contrast to overly complicated code/design, which is proven to be a factor of failure in a number of projects (e.g. Etherium and DAO's hack).
	Formal requirements/features management practices	*Koch et al. (2003) recommend that community discussions on requirements and targets of further development should be preceded by general preparations by the core team, incl. roadmap development.*
User-centered development	**Ease of adoption/fabrication**	*As there is no requirement to create a commercially viable product per se, open source projects tend to evolve more in line with developers' wishes than with the needs, existing environments and practical limitations of end users. No matter the quality, accompanying traits such as high set-up and usage costs might drastically harm adoption.*
	Placing an emphasis on internationalization and localization	*The number of translated languages (along with the number of downloads in the previous releases, and the number of developers) have been found to have positive impact on project popularity (Midha, 2007).*
	Usability and user experience	*One of the key advantages of open source in general is that users are not "locked into" using a particular vendor's system that only work with their other systems (Bridge, 2013). This, in addition to the fact that you can modify and adapt open source products for your own needs, should make open source products quite temptive and easy to adopt, which is not always the case, due to lack of focus on usability testing and user experience planning and execution of continuous improvements.*
	User-driven innovation through forks	*A fork cannot be perceived as something negative per se – only when it is a result of a conflict between coalitions within the original community that have not been resolved, it is definitely a strong sign of miss-management. In many other cases, forks power user-driven innovation without splitting the community. They may even be considered a corenerstone of open source development success and rising popularity in times of transit away from mass production models.*
Dispute Resolution	**Enforcing a contributor (license) agreement**	*Involving the community too early without a common core to organize members around, can create confusion, ambiguity, scope creep, and eventually project failure. One such common core might be the initial designs, even versions of the product being developed, but conceptually the license agreement is the core to look for – without one, resolving disputes in legal terms becomes an extreme burden.*
	Predictability through quality-oriented development	*It is hard to point out an area of work in which strong quality assurance/testing is not considered to be an important success factor, but given that the "Linus' Law" (Raymond, 1999) is marketed as one of the key competitive advantages of open source ("With a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone"), failutre to ensure/secure product of quality can undermine the sole foundations of the open source model. Therefore malicious practices such as 'code dumping' and ambiguous bug reporting, among others, are to be avoided, and continuous testing should be held to highest standards.*

	Open technical steering committee meetings	*As end users and active members of open source projects are usually spread all over the world, they need a central "place of appointment". The more open the steering committee meetings are, the more inviting they become for the entirety of the community's member base.*
Strong collaborative development environment / infrastructure	**Comprehensive collaborative development tool(s)**	*It is a commonly accepted fact that early versions of Linux were to some extend too difficult for nontechnical users to install and use, which may have negatively impacted the open source adoption curve far beyond this particular project. Comprehensive collaborative development tools are one of they ways to reduce complexity, given the fact that beyond code, they provide an opportunity to exchange support and preserve valuable information and resources, related to the project. This is of expeciall importance to hardware OS projects and it benefits equally end-users and contributors themselves.*
	Differentiate releases	*Speed ('release early – release often') is not the only thing to focus on, regarding releases - a clear declaration and identification of beta and stable releases is a must (Koch et al., 2003). It is part of the broader subject of managing the complexity of the project, given the multitude of developers and the greater number of versions to be maintained due to the freedom to modify the product (Helander, 2007).*
	Automate where possible (build, test, and continuous integration systems)	*Given the distribued environment, and open nature of OS projects, automation (in all of its aspects) has some particular advantages when it comes to open source. Analyzing 34 544 opn source projects on GitHub, Hilton et al. (2016) foud out that compared to projects that do not use CI, projects that use CI: (i) release twice as often, (ii) accept pull requests faster, and (iii) have developers who are less worried about breaking the build. Therefore, it should come as no surprise that 70% of the most popular projects on GitHub heavily use CI.*
	Encourage cost-effective reuse	*Importance of cost-effective reuse of contributions cannot be defined as something especially restricted to open source. In fact, if you're building software today, without continuously improving builds and tests, based on reusing the available contributions, and without using continuous integration, chances are you are not considered „a professional" any longer. The same goes for non-coding contributions.*
Inclusive development process	**Well-maintained technical documentation**	*A comprehensive documentation of the code, along with useful comments within code (Martin, 2015), are among the constituting elements of a well technically documented project. In the case of open hardware this would include standards - definition of open source hardware requirements. For some open hardware projects, the documentation may be the singe asset to collaborate on, hence it is critical to leverage feature-rich solutions that enhace collaboration to a maximum extend. Some propriatory tools have special offers for open source teams and are worth considering (e.g. "Dozuki" platform).*
	Contribution enablement	*Guide, online tool to search, tagging opportunities.* *Reputation management on the side of the contributors.*

	Contributors' code of conduct	*Implementing a code of conduct is seen from some as a tool to tackle the lack of diversity in open source, characterized with dramatically low representation by women, people of color, and other marginalized populations (Ehmke, 2014).*
	Providing timely and supportive feedback to contirbutors	*Project-related ideas and viewpoints of others in any respects must be accepted, appreciated and incorporated, if appropriate, but when it comes to contributtions to code/design, bug-fixing, and other technical contributions, given that those are at the top of the spear of an open source project, it is crucial that there is predictability, openness, transparency and fast response times for the communication and information exchange activities, related to those, with a high topicality and quality of contents (Koch et al., 2003).*
Consideration of the development life cycle	**Effort in all areas of system development**	*IT projects do not end with the release of a product, and in the case of open source, it is quite the contrary – a release is a simply of a new cycle.*
	Planning, design and development work upon introduction / initiation	*Some researchers (Moody, 2001; West, O'Mahony, 2005) point out that it is important for members to be involved early in the design phase, giving Linus Torvalds as an example, as he relied on others to design and implement crucial networking libraries (forming the so-called "lieutenant" system of senior technical leaders in the Linux kernel project), but as a matter of fact it was only a few weeks before the intial release of the project 0.01v. when Torvalds invited people to contribute with ideas on comp.os.minix (1991). Next to that, in a modular open source environment, work on design happens iteratively over the lifecycle, so most of the researchers do not identify it as a separate phase on its own (Saini & Kaur, 2014), assuming the pre-introduction effort.*
	User enablement and (client) support	*While it would be of help to increase the technical sophistication of the non-technical user and client community, it is also important to heighten the sensitivity of project teams (Ankolekar et al.,2004) and undertake such efforts (incl. through commercial partnerships), which would satisfy the expectations of general users. The opposite would harm adoption rates and prospects of success – Hai-Jew (2013) quotes clients' study, showing that the lack of professional (support) services is the top reason for preffering propriatory over open source software.*
	Being aware of the socio-technical evolution of the project	*No effective stakeholder strategy is possible to be applied if there's no clarity on a few important community and development-oriented subjects, such as the evolution of community's structure, what are project's contribution and maintenance policies, etc. For those to become clear, we need a mimimum of governance and decision-making arrangements, even for the smallest of projects, monitored and updated accordingly with the time passing.*