Modeling the Effects of Project Management Strategies on Long-Term Product Knowledge

Höst, Martin

Published in:
Product-Focused Software Process Improvement/Lecture Notes in Computer Science

DOI:
10.1007/978-3-642-31063-8_9

2012

Citation for published version (APA):
Höst, M. (2012). Modeling the Effects of Project Management Strategies on Long-Term Product Knowledge. In O. Dieste, A. Jedlitschka, & N. Juristo (Eds.), *Product-Focused Software Process Improvement/Lecture Notes in Computer Science* (Vol. 7343, pp. 104-115). Springer. https://doi.org/10.1007/978-3-642-31063-8_9

Total number of authors:
1

Modeling the Effects of Project Management Strategies on Long-Term Product Knowledge

Martin Höst

Department of Computer Science
Lund University
Sweden
martin.host@cs.lth.se

Abstract. In a team, people sometimes leave the team and become replaced by new persons with less experience, and sometimes people participate in new activities and thereby obtain new knowledge. Different processes, in terms of different management strategies, can be followed, e.g., to introduce people to new tasks so they get new knowledge. There is a need to investigate the long term effects of different strategies on a team's software product knowledge. This paper presents an initial approach for how this type of knowledge can be modeled as a stochastic process. Metrics representing the long term effects on knowledge are derived, and two different example strategies are investigated numerically. Based on this it is discussed how the model can be further elaborated and evaluated.

Keywords: software process modeling, product knowledge, learning, truck factor

1 Introduction

The relation between the software process and the product is of interest in order to be able to define a suitable process under a certain condition. In this paper the effects of the process in terms of how people are assigned to different tasks are investigated with respect to a team's long term knowledge of the product.

In software maintenance and evolution, teams are responsible for working with a rather large amount of software. The team is composed of a number of different engineers with different competencies, and the software consists of different modules, which means that different engineers can have knowledge about different modules. It is, of course, positive if many engineers have experience of many modules and if there for every module is several engineers that have experience of it.

In many cases there may be only few persons with experience of some modules even if there in an initial phase were several persons with knowledge about every module. Reasons for this might be that people leave the team. That is, the number of people with experience of a certain module is changing according to a dynamic process. A situation that is not attractable is when there are very

in proceedings of International Conference on Product Focused Software Development and Process Improvement (PROFES), 13-15 June, Madrid, Spain, pp. 104-115, 2012. The original publication is available at www.springerlink.com

few people with experience of a module, or when there are no people at all with experience of it. Staffing and assigning tasks to engineers is a typical task for management which can affect the knowledge of people in the team. People working with a module gain experience from working with it, but it is in many cases less expensive to have a person who already have experience of the module working on it.

One reason for working in groups with a shared responsibility of the product is that the knowledge about the code can be spread among several engineers, and the risk that certain parts of the code is only known by a single engineer is lowered. The term “collective code ownership” was coined as part of agile software development, meaning that anyone can change any code anywhere in the system at any time [1]. It is argued that this means that knowledge about the code is spread among the engineers and that code that is not well structured or too complex will be improved since many people are working with it.

The overall objective of this paper is to present a model for how the dynamic changes of knowledge about a module in a team can be modeled, taking into account that people sometimes leave the team and experience is lost and that people sometimes gain new knowledge. The objective is also to illustrate how the model can be used by showing the results for a number of example situations.

The outline of this paper is as follows. In Section 2 related work is presented and in Section 3 the analysis model is presented. The model is presented for two example strategies. In Section 4 the numerical results of the model for the two example strategies are presented. The results of using the model is discussed in Section 5, and conclusions and further research are presented in Section 6.

2 Related work

Ricca and Marchetto [2] investigated if there are “heroes” in empirical studies of open source projects, where they with the term heroes mean people who are the only ones who have contributed to different files. In a set of studied open source projects it was found that heroes are common in open source projects, and that they are faster in development than non-heroes.

In their work on formulating approaches for analysis of process compliance, Zazworka et al. [3] defined and studied a metric of “truck factor”. They evaluate the approach for analysis of non conformance in a case study on XP, where non conformance can be detected through a metric for truck factor. They define the truck factor as the maximum number of developers that could leave the team, and still at least a certain percentage of the modules would be known by at least a certain number of people. That is, they define the metric as one single number for a whole project. The formula for calculating the truck factor they define is

$$tf_{x,c} = \max\{n \mid \mathrm{cov}_x(n) \geq c\} \tag{1}$$

where c is a percentage, and $\mathrm{cov}_x(n)$ is the coverage, defined as the percentage of components that would still be known by the developers if n developers were

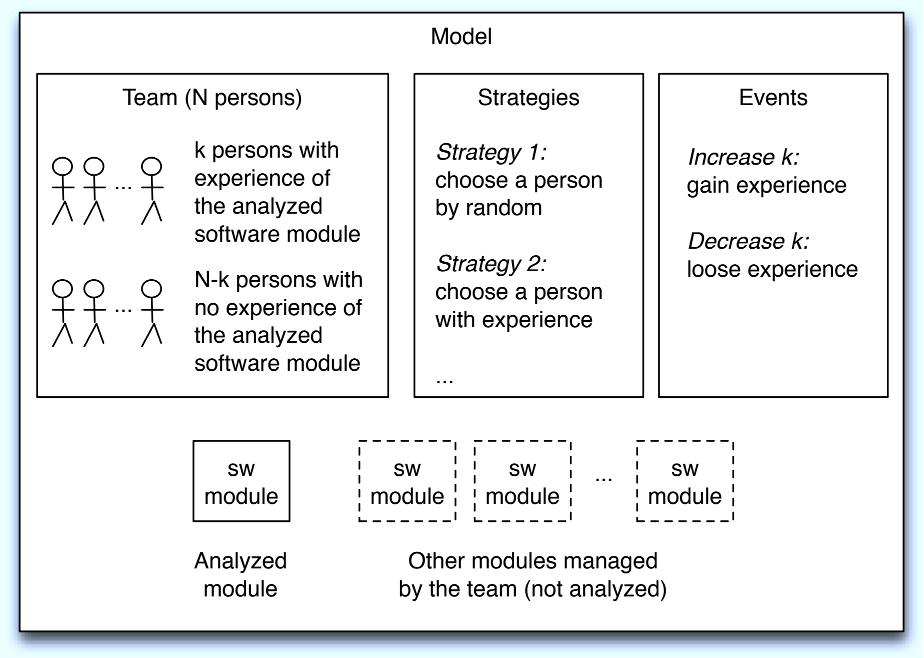
Model
Team (N persons)
k persons with experience of the analyzed software module
N-k persons with no experience of the analyzed software module
...
Strategies
Strategy 1: choose a person by random
Strategy 2: choose a person with experience
...
Events
Increase k: gain experience
Decrease k: loose experience
sw module
Analyzed module
Other modules managed by the team (not analyzed)

Fig. 1. Overview of the model

absent according to different types of situations, x, as worst case, average case, or best case.

Torchiano et al. [4] build on the work by Zazworka et al. [3] and investigate threshold values for the truck factor, and they look at data from a set of 20 open source projects. Ricca et al. [5] continue this work and present a tool for calculating the truck factor and investigate the sensibility of the truck factor metric. They conclude that the metric is sensible, but that more research is needed on how it is calculated and that there may be scaling problems for large projects.

Compared to related work on knowledge about software in teams, the model presented in this paper focuses more on the dynamic changes of knowledge than existing models. This is done at the cost of focusing on only a single module at a time of the software, instead of at the whole software system.

3 Model

3.1 General overview

The model is outlined in Figure 1. It is based on the following assumptions:

- The model describes the experience of one code module in the product. The size or type of the code module is not described by the model, but the basic idea is that a person in the team either has experience and knowledge about the module or not.

- The team consists of N engineers. That is, the team size is constant. When one person leaves the team, that person is replaced by a new person.
- When a new person is introduced in the team, that person does not have experience of the code module.
- When a person in the team gets an assignment to work with the code module it is assumed that the person during this work obtains experience and knowledge about the module. When this happens it can be assumed that the person obtains experience. If there are other people in the team who have experience of the code it is probably easier for a novice to work with the code since there are other people to ask for example when the documentation is hard to understand.
- In the team there are k persons that have experience and knowledge about the code module under study. k can take any value from 0 to N and is used to model how the knowledge changes over time.
- How people are assigned to maintenance tasks are decided by the strategy of the management and the team. One strategy could be to always let a person with experience of the module work with the task, and another strategy could be to always let a person with no experience of the module work with the task. The strategy affects how how often people in the team get new experience of the module under study.

It is assumed that new maintenance jobs arrive to the organization with intensity λ. The intensity of people leaving the team and thereby being replaced by someone else without any knowledge about the module is μ. The time between new assignments is exponentially distributed with mean value $1/\lambda$, and the time between people leaving the team is exponentially distributed with mean value $1/\mu$. Define ρ as $\rho = \lambda/\mu$.

Altogether this means that the model describes how k varies between 0 and N over time. This is modeled as a stochastic Markov process with discrete levels in continuous time (e.g. [6]). This type of model is used to model a wide spectrum of processes and systems in the literature, e.g. in different areas like performance behavior in telecommunication systems and user behavior in software testing. They provide a well known and easy way to derive mathematical expressions, and on the same time the possibility to model a level of detail that is sufficient in many situations. The Markov model that is used in this paper is outlined in Figure 2. This model shows that there are transition intensities between adjacent states, and the states represent how many people in the team that have experience of the module. In this model, λ_k denotes the transition rate from state k to state $k+1$, and μ_k the transition rate from state k to state $k-1$.

It is, of course, possible to define more complex models with transition rates between more than adjacent states, e.g. if two novice programmers work together with a module. However, for the strategies presented in this paper the model presented in Figure 2 is enough, and there is no difference between this model and a more complex model when it comes to how the metrics presented below are calculated.

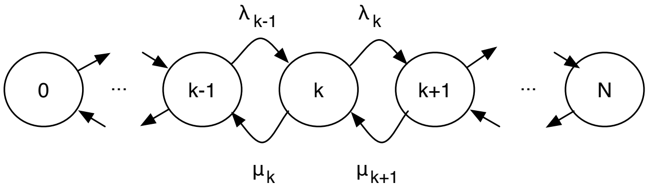

Fig. 2. Markov model

The number of people in the team with experience can be affected by management through applying different *strategies*. Strategies may concern both how people who not yet have experience can get experience through involvement in projects and in other ways affecting how people leave the team and how they get new experience. The transition rates between different states (i.e. λ_k and μ_k) can then be estimated based on the strategies. In this paper two different example models based on two different strategies are presented. In the example strategies presented below it is shown how the transition rates can be decided from the strategies.

The fact that knowledge is gained by working with the code module deserves a discussion. There are, of course, other ways to obtain knowledge, such as through training, participating in inspections, or other ways. However, contributing to the code has been used as an indication of knowledge in other studies ([2–4]), and it is at least an intuitive indication of knowledge. Fritz et al. [7] investigated the relationship between activity for a piece of code and knowledge about it and found that activity could be used as an indicator, although additional factors could also be considered such as role and professional work experience. Of course, all tasks are not large enough to give experience enough for a person to obtain knowledge. As part of using the model the type of task that is assumed to give experience must be defined. A factor that is not considered in the model presented in this paper is the fact that people forget their knowledge after some time and that knowledge about a module may be outdated after some a longer time when a module has been extensively modified. This is a result of a trade-off between having a simple and a complete and detailed model.

The model that is presented is analytical, but it would of course also be possible to define a simulation model for this purpose. The advantage of an analytical model is that it is possible to investigate more values for different parameters in a certain amount of time, and when formulas can be derived they ca give a knowledge as such of the relations between different factors. Simulation-based analyses require longer execution times, although they make it possible to analyze more complex processes. In this research we have seen that it is possible to define an analytical model which answers the basic questions that are of interest.

3.2 Metrics based on the model

The goal of the analysis is to investigate which strategy helps maintaining knowledge about each code module as long as possible. Then, we want to know the answer to questions about the risk of loosing all people with that knowledge, and how long it would take to come into that situation. Based on this, two metrics are investigated in this paper:

p_0: The probability that the team at a random point in time will be in the state that no one has knowledge about the module.

t_i : If there are i persons in the team with experience of the module, t_i is the mean time it takes until the first time that no one has experience of the module.

These metrics can be derived and calculated from the model as follows. Let $(q_{ij}) = Q$ denote the $(N+1) \times (N+1)$ transition rate matrix for the process with $N+1$ states $(0, 1, \ldots N)$. Let the rates in the first row denote the rates from state 0, the rates in the next row denote the rates from state 1, etc, i.e. a traditional transition rate matrix as described e.g. by Cox and Miller [6].

The calculation of p_0 is based on calculating the steady state probabilities $p = (p_0, p_1, \ldots p_N)$, that is the probabilities that the process will conform to after an infinite amount of time. We use this as an indication of the state probability at a random point in time. The probabilities p can be found by solving

$$0 = pQ \tag{2}$$

with the normalization condition $\sum_i p_i = 1$ (see e.g. [6]).

In order to determine t_i, state 0 is defined as absorbing, by adapting the Q-matrix by setting the first row to zeros:

$$Q = \begin{pmatrix} 0 & 0 \\ \hat{q_0} & \hat{Q} \end{pmatrix}, \tag{3}$$

i.e. there are no transition rates from state 0, otherwise Q is not changed. This means that $\hat{q_0}$ is a vector with the transition rates to state 0 and $\hat{Q}$ is an $N \times N$ transition rate matrix for the transitions between states 1 to N. Then the mean times t_{ij} that the process stays in state j before absorption if it starts in state i can be derived as

$$(t_{ij}) = T = -\hat{Q}^{-1} \quad \begin{array}{c} 1 \leq i \leq N \\ 1 \leq j \leq N \end{array} \tag{4}$$

A proof of this is presented by Tavere [8]. Actually that proof is presented for two absorbing barriers, but it easily follows that it is valid also for one absorbing barrier as in this case. From this it follows that the mean time until the process reaches state 0 if it is in state i is

$$t_i = \sum_{j=1}^{N} t_{ij} \quad 1 \leq i \leq N \tag{5}$$

That is, in order to calculate the metrics (p_0 and t_i) the following factors must be known, measured or estimated: λ (i.e. how often new maintenance tasks for the module arrives), μ (i.e. how often people leave the team), and what strategy is applied.

It should be noted that it is easy to define other metrics in the same way as p_0 and t_i. For example, a measure of few (≤ 1) persons with experience could be defined as $p_0 + p_1$.

3.3 Strategies

Two strategies for how people are assigned to tasks are defined and analyzed in this paper.

- *Strategy 1:* A person is chosen for the task by random. That is, all persons in the team are equally likely to work with the task.
- *Strategy 2:* If there are persons in the team with knowledge about the task, one of them will be chosen. Only if no one has experience of the task, someone without experience will be chosen.

The two strategies are further presented below.

Strategy 1 According to this strategy, every time a new job arrives a person in the team is chosen by random. This means that the probability that a person without experience of the module is chosen in state k is $(N-k)/N$, which means that the intensity of an increasing the number of people with experience is

$$\lambda_k = \frac{N-k}{N}\lambda \qquad (6)$$

When people leaves the team and are replaced by someone with no experience then the probability that the person had experience of the module is k/N. That is,

$$\mu_k = \frac{k}{N}\mu \qquad (7)$$

The state probabilities can be found through equation (2), but it is also possible to decide an expression for p_0 as (see Appendix A.1):

$$p_0 = 1/(1+\rho)^N \qquad (8)$$

For this strategy, formula (8) can be used instead of formula (2), which is valid for any strategy.

Strategy 2 This strategy is to always choose a person who has experience of the code if such person is working in the team. This means that the intensity of people with experience leaving the team is the same as in model 1, i.e. formula (7), but the intensity of acquiring experience is

$$\lambda_k = \begin{cases} \lambda,\ k = 0; \\ 0,\ 1 \leq k \leq N; \end{cases} \qquad (9)$$

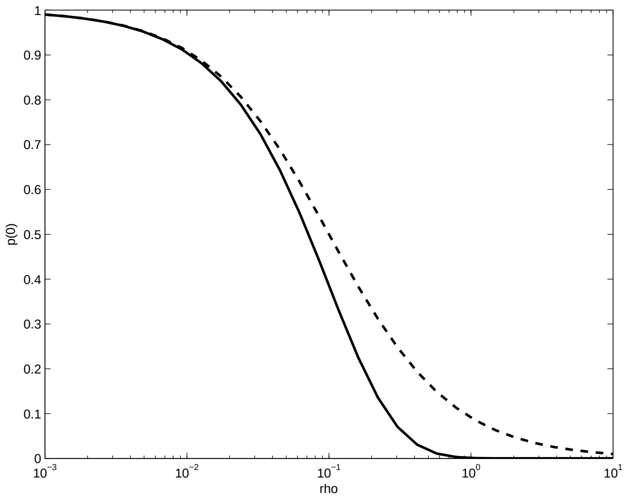

Fig. 3. p_0 as function of ρ with $N = 10$. The solid curve is for strategy 1 and the dashed curve is for strategy 2.

Note that when calculating the stationary probabilities, p, only p_0 and p_1 will have values larger than zero for this strategy. It can easily be shown that the p_0-probability is (see Appendix A.2)

$$p_0 = 1/(1 + N\rho) \tag{10}$$

In the same way as in strategy 1, formula (10) can be used instead of formula (2), which is valid for any strategy.

4 Model results

In this section it is shown what the results of the model are for a set of different values. In Figure 3 it is shown how p_0 varies for different values of ρ with the two strategies. The solid line shows strategy 1 and the dashed line strategy 2. Note that p_0 depend only on the relation between λ and μ, i.e. ρ. For example, if $\lambda = 1$ month^{-1} and $\mu = 0.1$ month^{-1} this means that $\rho = 10$.

Figure 4 shows how t_i varies for different values of i. The three solid curves show the results for three different values of ρ for strategy 1. The dashed curve shows the results for strategy 2. Note that for strategy 2 the time to absorption is independent of λ, which is why only one curve for strategy 2 is shown.

As expected, strategy 1 is better than strategy 2 when it comes to the investigated metrics. For example, comparing the two curves in Figure 3 for $\rho = 1$, p_0 is very small for strategy 1 ($1/2^{10}$), while it is quite significant for strategy 2 ($1/11$).

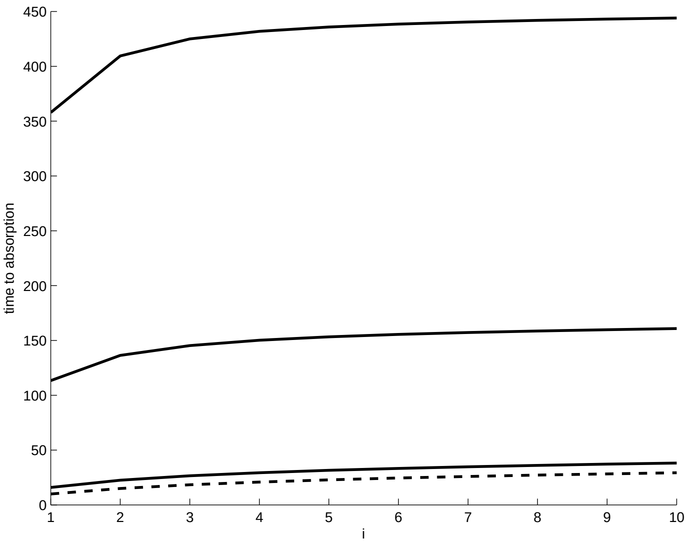

Fig. 4. t_i for $\mu = 1$ and $N = 10$. The three solid curves show the results for strategy 1, where $\rho = 0.1$ in the lower curve, $\rho = 0.5$ in the middle curve, and $\rho = 0.75$ in the upper curve. The dashed curve shows the results for strategy 2.

5 Discussion

A model like this can not be expected to present values that perfectly correspond to actual values. One reason is that the input parameters can not be known in detail. Instead, estimates of values must be used. It is the same for the definition of strategy. In most cases it is not possible to formulate exactly what strategy is used, only to describe what overall strategy that is most similar to the actual strategy that is likely to be followed in a team. It is also worth noting that it is not probable that the results will be valid when very long times are predicted, e.g. as in the upper curve in Figure 4 where times longer than 400 months are predicted if, for example, $1/\mu = 1$ month. This type of result should more be seen as the model does not predict any problem with respect to knowledge over time. It should also be noted that, since the result of the model is the result of many events in a stochastic process, the variation of the results may be large. Especially the actual variation around the mean value t_i may be large. To illustrate how large the variations can be, a team with with strategy 1, $N = 10$, $\lambda = 0.75$, and $\mu = 1$ has been simulated 200 times with a discrete event simulator. The resulting measures of t_5 are illustrated in the histogram in Figure 5. It can be seen that the variations are large. When the results of this kind of model are used it should be known that the actual results in reality will not be exactly as the mean value is predicted.

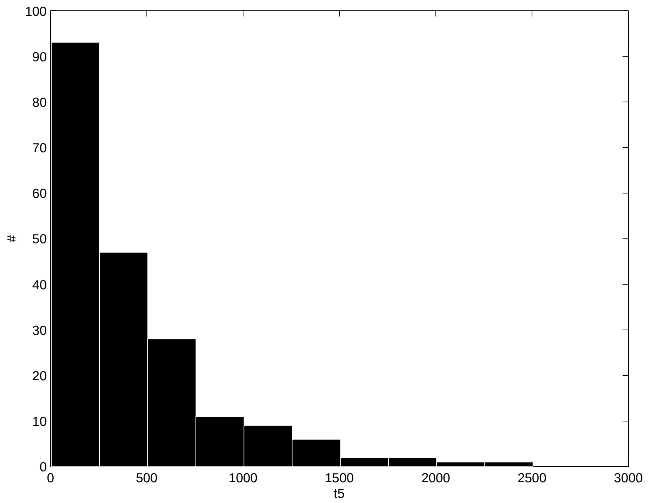

Fig. 5. Simulation results, where a team with $N = 10$, $\lambda = 0.75$ and $\mu = 1$ is simulated and the time t_5 is measured 200 times.

An assumption in the model concerns the distribution of times between assignments and the times between people leaving the team. The model assumes that these times are exponentially distributed with different mean values. In reality it is of course not certain or even likely that this is true. However, in order to derive a model certain assumptions and simplifications of the reality must be made and the exponential distribution do not have to bee too far from reality. There are other aspects that the specific distribution that can be discussed, such as if events are independent o not. A realistic situation that is not modeled is that an external event, such as the start of another large project, could mean that several people left the team simultaneously. When models are formulated, simplifications must be made, and it is important to be aware of this when the results are used. We believe that the model represent a reasonable trade-off between simplification and inclusion of details.

The realism of the two formulated strategies can also be discussed, although they should be seen more as examples of strategies than actual strategies. However, the two strategies can be seen as two basic ways of assigning people to tasks, where strategy 1 is more focused on assigning people to new tasks than strategy 2, which is intended to model a very "short sighted" management strategy. Concerning strategy 1, people are not assigned to tasks by random in reality. However, people in the team are in a real situation working with several different tasks, which means that it is not possible to assign any person to any task at any time. This means that someone else than would initially be seen as most suited for a task may have to be chosen. Even if this is not the same as strategy 1, it may be nearer to the reality than for example strategy 2.

The proposed usage of the model is not to use it to derive definite and exact figures of future results. It is more to follow a procedure where a process is modeled as good as possible with respect to parameters and strategy. This probably involves formulating a strategy in the same way as was presented in Section 3, but it may also be possible to choose one of the two models that are presented in this paper. Based on this it is then possible to investigate different "what-if"-scenarios by changing parameters and/or strategy and to compare the results in order to get an understanding of the effects of different types of changes. That is, when processes and management strategies are improved, different alternatives can be evaluated with this type of model. For example, a manager responsible for a team can investigate the implications of different strategies on the product knowledge. Then the results of his model could, in combination with other factors, such as estimated cost of different strategies and the impact of delivery date, be used when processes and strategies are defined.

6 Conclusions and further research

The presented model can be used to get an indication of the long-term effect on knowledge in teams for different types of strategies. That is, it is one example of a model that can be used to analyze the effects of different processes on product. In this model the process aspect concerns the strategies for staffing in projects, and the product aspect is the team's knowledge of the product.

The model can likely be used when different strategies are compared and the likely effects of changing parameters are studied. It can be used as one source of information when different strategies are considered. Other sources of information can for example be estimates of the costs of different strategies. However, this should be seen as a first attempt to formulate a model, and a number of areas for further research remain.

There is a need to formulate and investigate more strategies than the two investigated in this paper. Strategies based on "pair-programming" [1] may for example be investigated. The model needs to be further evaluated by comparing the results of it to empirical measurements. This could, for example, be carried out in the form of surveys with real industrial projects. There is also a potential possibility to investigate knowledge about other factors than code modules, e.g. treating "system knowledge" as one unit of analysis.

A Appendix

A.1 Derivation of (8)

Equilibrium gives that

$$p_{k-1}\lambda_{k-1} = p_k\mu_k \qquad 0 < k \leq N \tag{11}$$

which is the same as

$$p_{k-1}\lambda\frac{N-(k-1)}{N} = p_k\mu\frac{k}{N} \tag{12}$$

that is

$$p_k = \rho \frac{N-(k-1)}{k} p_{k-1} = \rho^k \binom{N}{k} p_0 \tag{13}$$

The normalization condition means that

$$1 = \sum_{k=0}^{N} p_k = \sum_{k=0}^{N} \rho^k \binom{N}{k} p_0 = (1+\rho)^N p_0 \tag{14}$$

which means that $p_0 = 1/(1+\rho)^N$.

A.2 Derivation of (10)

Since $\lambda_k = 0$ for $k > 0$, equilibrium means

$$p_0 \lambda = p_1 \mu \frac{1}{N} \tag{15}$$

which means that

$$p_1 = N \rho p_0 \tag{16}$$

Since the sum of p_0 and p_1 is 1, $p_0 = 1/(1 + N\rho)$.

Acknowledgment

This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering, (`http://ease.cs.lth.se`).

References

1. Beck, K.: Extreme Programming Explained. Addison Wesley (2000)
2. Ricca, F., Marchetto, A.: Heroes in FLOSS projects: An explorative study. In: 17th Working Conference on Reverse Engineering. (2010) 155–159
3. Zazworka, N., Stapel, K., Knauss, E., Shull, F., Basili, V.R., Schneider, K.: Are developers complying with the process: An XP study. In: proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). (2010)
4. Torchiano, M., Ricca, F., Marchetto, A.: Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold. In: WETSoM. (2011)
5. Ricca, F., Marchetto, A., Torchiano, M.: On the difficulty of computing the truck factor. In: Proceedings of 12th International Conference on Product-Focused oftware Process Improvement. (2011) 337 – 351
6. Cox, D., Miller, D.: The Theory of Stochastic Processes. Chapman & Hall (1965)
7. Fritz, T., Murphy, G.C., Hill, E.: Does a programmer's activity indicate knowledge of code? In: Proceedings of the the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC-FSE. (2007) 341 – 350
8. Tavere, S.: A note on finite homogeneous continouos-time Markov chains. Biometrics **35**(4) (1979) 831–834