A. Slyva, Student
V. Spivachuk., PhD in Engr., As. Prof., research advisor
Khmelnytsky National University

10 FACTS ABOUT PROGRAMMING YOU PROBABLY DID NOT KNOW

The task of programming is becoming increasingly common, but there are still many facts that people do not know about programmers and programming itself. This post features 10 little known facts about programming. Like other intellectual activities, the task of programming and how people learn to program computers is well studied. In fact, with more and more people learning to program regardless of language, tool or platform used, it is natural that few people actually know about certain important facts about programming and software development. From an academic standpoint, the areas of software engineering and education bring forward several very interesting studies obtained by experiments whose results are presented in masters' and PhD theses. And based on these results I chose the facts mentioned in this report along with appropriate references. Based on this context, I will present 10 important facts that, unfortunately, are little known by whom programs. But before, a warning: these facts present results of experimental and empirical research that have specific contexts. What I mean is that there is some room for discussion of the applicability and generalization, but knowing what has already been discovered and studied is important and, at least, can instigate discussion and how close that information is to the reality of the reader.

Programmers have a tendency to report their problems incompletely [4].

This fact is related to Psychology field research. The results indicate that when a person has a problem he/she does not report complete information about the problem, especially when it is responsible directly or indirectly. This result has been confirmed experimentally with programmers and one of the main reasons is the following: to fully report a problem is seen as a sign of weakness that can lead to some kind of judgment of skill and proficiency by whoever is listening to the story. This situation is more common when it comes to a fundamental error committed by novices but such mistakes are also allowed by specialists.

Developers seek other forms of help before talking to coworkers [3].

The fact of communication with other people do not have priority when a programmer needs help again is related to the sense of judging of what other people do when they know the difficulty. However, a site like Stack Over flow [5] has flourished exploring this type of behavior by aggregating help in various aspects of communities for developers. This is a rather sad trend that shows that programmers are ready to trust strangers on the Internet but are not ready to trust their colleagues and mentors.

Progress in programming can be classified into 4 stages.

The classification of a programmer progress is important to support multiple metrics involved in software development and also help project managers and other professionals to evaluate how good the project is as a whole.

Moreover, it is also important to know in which phase of the progress the developer is to, among other things, offer some kind of help so that he does not spend too much time stuck in a specific task to the point of delaying any deliveries. An interesting classification identified (automatically) four possible states of progress: Complex Programming b) Making Progress c) Slow Progress d) Stuck.

Developers slow to ask for help when facing problems.

This is related to the way people learn how to program; basically, the act of teaching follows the line of learning Mathematics: a little theory, one or two examples and many exercises. This format takes learners to try hard on exercises and, quite often, to solve everything themselves without asking for help. This attitude is not bad and is even recommended, but you need to know to what extent should stop trying and ask for some form of help. Large companies often mention the problem that some employees do not report their problems because of fear of bosses and possible removal from large projects

Developers find beatable and unbeatable barriers.

This may seem obvious, but it is very important to be detected, since a programming barrier can lead to serious term, team morale and confidence problems. One of the main difficulties of detecting barriers and classify them is the fact that this information may be subjective. In other words, asking directly to the programmer if he/she is with some beatable or unbeatable barrier already affects the result, as it can not always be sincere. There are also some implications in terms of ego and moral just by identifying this type of barrier on programming.

Programmers spend approximately 30% of the time surfing the source code [1].

People who program know that most of the time relies on a editing source code tool. However, how time is divided between the editing tasks remains unclear from the scientific point of view. According to an important study, it was found that approximately 30% of a programmer working time is not spent editing the text (by including, editing or deleting), but surfing between multiple files along the source code. The navigation involves research, observation, information gathering, memorizing and other activities. That is, you could say that programming is an activity whose third part is just contemplative.

Remote programmer productivity is lower than the productivity of local programmers.

This claim about productivity is controversial, especially when routines such as home office, remote working and global software development projects had become increasingly high. Anyway, there are concrete evidences based on several metrics of software that, in fact, remote programmers do not produce as much as programmers working together in the same place.

But it does make sense to think this way if we analyze the other facts of this list, for example, the preference for the lack of communication with other people. In fact, informal communication is a major factor that influenced the results of this research, because asking that hint in the meeting during a coffee break is very important according to what was found alone.

The main error messages, execution times and runtime compilation errors and the average time to solve them [2].

Error messages are very specific to each language problems and runtime compilation errors. To highlight some cases mention the master's thesis of Suzanne Marie Thompson, as she looked at a lot of Java programmers in different scenarios and collected many interesting facts about them. The tables below include a bit of history about errors and the average time to correct them.

Although the study focus on a very specific context (learning the Java language) is possible to make a comparison with other scenarios and situations and prove that much of the most common errors occur in different contexts.

The software maintenance consumes more than 50% of the effort.

Software maintenance involves the manipulation of legacy code. There is a study about effort that shows as a result that the division is not equal between creation and maintenance. In the study that mention a value of more than 50% of the effort due to software maintenance there is also a great discussion on software evolution towards its maintenance and the necessary tasks for both. Surely is worth taking a look at this reference before making that decision about starting to develop the solution from scratch or working with an existing code base.

The software maintenance consumes between 40% and 90% of costs.

One of the main rules of business people says it is much more expensive to get a new customer than to keep an existing customer. However, according to software engineering researches, the reality is somewhat different when it comes to code: to keep the code running through maintenance tasks can cost up to 90% of all project costs. These statistics are very general and were obtained in a very particular context of the 487 organizations studied for this research (which is from 1980). Certainly there are many factors to consider, but at least there is a starting point for analyzing courses and discusses this topic when talking about software maintenance.

Peer code review can discover up to 60% of bugs.

Code review made by other people, either in the form of pair programming or not, is really effective. There are many studies on this, but one of the key of them indicates that up to 60% of bugs can be discovered (but not necessarily fixed) when more than one person reviews the source code. This study is relatively old and can be said that it is one of the key influencers of techniques involving agile process and other ways of developing software whose main focus is on activities, steps, organization and other skills not as technical as programming.

REFERENCES

1. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks [Електронний ресурс] // ieee.org. – 2006. – Режим доступу до ресурсу: https: // ieeexplore.ieee.org/ document /4016573?tp=&arnumber=4016573&url=http:%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4016573.

2. C.F. K. An Exploratory Study of Novice Programming Experiences and Errors [Електронний ресурс] / Kemerer C.F. // researchgate.net. – 2014. – Режим доступу до ресурсу: https: // www.researchgate.net/publication/ 240762269 _An_Exploratory_Study _of_Novice _Programming_Experiences_and_Errors.

3. DLaToza T. Maintaining Mental Models: A Study of Developer Work Habits [Електронний ресурс] / T. DLaToza, G. Venolia, R. DeLine // IEEE. – 2016. – Режим доступу до ресурсу: https: // cs.gmu.edu/~tlatoza/ papers/icse2006.pdf.

4. ShraugerJ. The relative accuracy of self-predictions and judgments by others in psychological assessment. [Електронний ресурс] / S. J, T. M // https://psycnet.apa.org.

– 2016. – Режим доступу до ресурсу: https://psycnet.apa.org/doiLanding?doi=10.1037%2F0033-2909.90.2.322.

5. Top Questions [Електронний ресурс] // stackoverflow.com – Режим доступу до ресурсу: https://stackoverflow.com.