The Hidden Job Requirements for a Software Engineer

Cristina Marinovici
Pacific Northwest National Laboratory, Richland, WA
cristina.marinovici@pnnl.gov

Harold Kirkham
Pacific Northwest National Laboratory, Richland, WA
harold.kirkham@pnnl.gov

Kevin Glass
Pacific Northwest National Laboratory, Richland, WA
kevin.glass@pnnl.gov

Abstract

In a world increasingly operated by computers, where innovation depends on software, the software engineer's role is changing continuously and gaining new dimensions. In commercial software development as well as scientific research environments, the way software developers are perceived is changing, because they are more important to the business than ever before. Nowadays, their job requires skills extending beyond the regular job description posted by HR, and more is expected. To advance and thrive in their new roles, the software engineers must embrace change, and practice the themes of the new era (integration, collaboration and optimization). The challenges may be somehow intimidating for freshly graduated software engineers. Through this paper the authors hope to set them on a path for success, by helping them relinquish their fear of the unknown.

1. Introduction

Technology evolution creates a new and exciting reality, where software is used everywhere and in everything. Software employment in every small thing is changing the role of the software developer and at the same time the expectations people have of developers. Just having the technical skills is not enough any more. The days when developers were working in isolation, focusing on a single task are becoming history. Today, in some fields, the necessity to deal with a massive volume of data is changing the developer role, making him/her more important for the business. In other fields, developers become testers and act for the end users. Others may interact more and more with customers. Moreover, developers have to be project managers, business analysts, security experts and even domain experts. The developers facilitate, yield, and require more fundamental scientific understanding of the domain that they work on, therefore they must become domain experts. This is the environment in which the freshly graduated software engineer is supposed be prepared to work.

Usually, a software developer job description presents only the quantifiable aspects of the work and cannot account for the dynamic on-the-job aspects. It is easy to decide whether or not the developer possesses the domain knowledge and the skill set. But there are other factors defining what is expected of the (employed) software developer. These factors relate not only to the necessary dedicated skill set and the level of domain knowledge, but also to the ability to communicate in and outside the team, to adapt to new technologies and tolerate ambiguities.

For a software engineer working in a scientific research community, the employer's expectations are driven by different aspects compared to the software engineer working in a commercial software development environment. Much of what is taught in schools focuses on business-oriented design used to develop commercial software. In the commercial world, software requirements are gathered to define not just the desired project, but to also understand future directions. This allows the developers to design defensively, create maintainable, reliable and extensible code.

However, this is not a typical development environment for a science and engineering software project. Further, software design from an academic perspective is radically different then the reality of scientific software development.

The science and engineering oriented development environment is more dynamic. Though the purpose of the project should remain consistent through its lifetime, the underlying algorithms, design and numerical approaches change constantly. That may mean that high-level requirements are more constant than low-level ones. Sometimes even high-level requirements could undergo several adjustments. For scientific software projects, this entails a constant shift to unpredictable courses that inhibit the ability to design for future directions. An environment like this is not amenable for gathering clear and constant software requirements. In many cases, the programming strategy

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.611

focuses on scientist's personal theories, with little attention to the broader software project, making code difficult to extend.

These are only some of the hurdles that make scientific software development hard to maintain, extend and susceptible to being unreliable.

Lately, the scientific software community started to adopt methodologies and techniques that proved to enhance productivity in commercial software development. This paper shares the authors' experience working on scientific research projects, and extrapolates the findings with the hope of inspiring others striving to become better software engineers.

2. Skill set

The most important and desired skill from any engineer is the capability to "properly" design a system. It is a matter of having a global view of the problem, while accounting for future potentials. What today is really high-tech and expensive, tomorrow will be commonly used.

For example, the task of storing real data becomes cheap and easy, being facilitated by the creation of large data warehouses in which the information is kept. Therefore, what is left as the most challenging task is properly analyzing the data. *"*We're still in danger of suffering from shortsightedness when it comes to data custodianship. Every experiment needs a clear plan in place to ensure that a record of the original observations is still available and readable, even decades into the future [1]."

The traditional academic curriculum, including research and project classes, influences the shaping of the student software engineer, preparing him/her for the real world work. Some of the key skills to possess originate from this academic training. These include:

- Use of scripting languages (i.e. UNIX shell scripting, Python) for extracting and accessing data from various sources. Taking advantage of increased computer speed, the scripting languages are becoming more and more important for the future applications [2].
- Ability to pre-process the real data and "clean" it (remove noise, errors, duplications and a variety of other defects), which are preliminary steps for analysis. Often thoroughness during this step yields the greatest benefits. It is well accepted that you can recover from bad analysis, but you cannot recover from bad data. A simple analysis of clean data can be more productive than a complex analysis of noisy and irregular data.
- Explore data - how to go from "the bits" to actual understanding of the data. Appropriate statistical methods (many scientists and engineers are surprisingly bad at statistical analysis), and educated exploratory data analysis can reveal great interesting trends in data [3].
- Ability to employ numerical linear algebra, computational and statistical methods to model data. These days, this may be as important as coding knowledge for a software engineer. Generating an output may be of no help without the benefits of interpreting it and understanding whether it makes sense or not.
- Competence in using visualization tools, which needs to be complemented most of the times by specific domain knowledge to interpret the data. This brings more fundamental scientific understanding and therefore influences the decision-making processes. In most scientific research projects, a picture (or a graph) is worth a thousand words [4].

It is really important to ensure that students get exposure to quality assurance (QA) and software testing skills and techniques. This does indeed prepare graduates with degrees in computer science for various careers in software engineering. In many product lifecycles, the testing is performed under pressure and therefore a good QA skill base is valuable and desired for a software engineer. Testing skills, that can be applied in any software engineering work-field, include the ability to :

- focus on what is really important and matters for the product delivery,
- actively identify, listen to, and learn from context drivers,
- dynamically prioritize testing, considering the risk-value and impact on functionality,
- create effective decision-making workflows ,
- avoid wasteful re-work in all aspects of test implementation and execution,
- accept mistakes as a way of learning and embrace risk as a way to progress [5].

Doubtless there are more.

In product development, projects do not always evolve as planned. The authors have presented detailed descriptions of the lessons learned from the development of a scientific research project in power system field area in a previous paper [5]. It is important to mention that the project was facing the de-scope decision. To bring back the project on track, as a first step, a "scaled-down" version, with fewer features was produced. Adoption of Agile methodologies with incremental releases (and reduction of featurecravings) also contributed. The teams started to communicate more, achieving quick and timely responsiveness and better collaboration. As a result, it has been noticed improved efficiency through better flow of the features from design, development, testing to integration. The team was able to see convergence of value, and most importantly reduction in bug count.

2.1. Domain Knowledge

Domain knowledge is hard to teach but very valuable. It requires practice and experience in the field, and is often the requirement that distinguishes software engineer qualifications. In scientific research groups, the software engineers grow their skill sets and domain knowledge by working on dedicated projects. They are able to develop a technical vocabulary, to understand the business and technical terms and, at the same time, to deal with the ambiguities of the scientific field. Ambiguities and misinterpretations of requirements when translated from business language into scientific language are two major factors introducing costly defects into product design. As the level of complexity of the product increases, the weaving between requirements and state of the software-product becomes more intricate [6].

2.2. Creativity

Besides really good domain knowledge, creativity is the driving factor of the software engineering profession. As Edward de Bono stated "creativity involves breaking out of established patterns in order to look at things in a different way [7]."

The necessity to incorporate rigor and subjectivity as well as creativity into the scientific process leaves space in every implementation for biases, and particular threats to validity [8]. Nevertheless, creativity has a direct impact on the business, as has been demonstrated by the big commercial software companies; workplaces where talent and innovation are unleashed create a more productive, efficient, and profitable environment. In the research community, searching for alternative explanations, finding an adequate software solution while accounting for the methods' limitations, and having a self-critical attitude are vital.

3. Adoption of Dynamic and Evolving Methodologies

The economic mechanisms of the market are forcing the software industries to move away from the long years of development efforts, and to compete with each other in releasing better products faster . Adoption of dynamic and evolving methodologies can unleash a world of possibilities, if applied properly.

It may not be obvious to the recently graduated software engineer, but *improper* application of dynamic methodologies and their trends may actually hinder creativity. Trying to develop a solution outside the carefully monitored framework may lead to disaster. The software engineer should keep an open mind, understand his or her role and responsibilities within the team framework, and just embrace the challenges.

Adoption of new methodologies creates psychological changes for the software engineer. The daily scrum framework increases team interaction, creating a camaraderie connection, and influencing team velocity. It is important to understand that the scrum is not a status review, it is a way of informing the team of the potential blockers and to plan on how to surpass them. Early tackling of blockers increases efficiency, improves code quality and may reduce defect counts.

The adoption of new methodologies (i.e., lean, Agile, XP, continuous delivery, etc.) changes the development process by asking more flexibility of the software engineers and their product. The change also affects the planning, forcing a better scope of the design and approach it through incremental steps.

The effect of new methodologies adoption is visible in the end-product. In each sprint, the code is implemented end to end, being tested and integrated right away; this fact makes the product almost ready for production. Every project is different, of course, and therefore there is no ideal sprint length. For our scientific power system software development project, the length of the sprint was correlated with project milestones and resources availability.

In any project, either scientific or commercial , the adoption of dynamic methodologies gives developers the possibility to experience different roles (scrum master, tester, user, developer, architect or manager). This change in roles provides a better perspective and understanding of the task(s), as well as a better estimation of the work-effort. In his article, Scott Sehlhorst describes estimation as "reflection of probabilities distribution and not specific numbers. Estimation carries a lot of risk. The estimates can be wrong, the estimates can be expensive, and the estimates can be reused. Risk is not necessarily a reason to avoid estimating …. Estimates can have value - informing rational top-down investment decisions, allowing for efficient investment of team effort by getting the most bang-for-the-buck, and providing motivation, feedback, and learning to team members [9].”

3.1. Distributed Teams

Having your team distributed all over the globe is not a new thing any more. Working in global teams has its own problems. The time difference is the major factor that influences team dynamics. Communication channels should nevertheless be maintained open; for the project’s success discussions should occur often. Working in a distributed team on the same product will alleviate the meaning of work hours; you will be required to work late in the night or bright and early in the morning to be able to communicate with your teammates in other parts of the world.

This is one of the cases where flexibility and adaptability of the software developer is really important. Communication that does not happen in the face-to-face manner will require the use of technology (i.e., conference calls, screen sharing, and so on).

Colocation of team members is a way of improving team interaction and allowing a better knowledge exchange. In the case of multi-disciplinary teams the knowledge exchange occurs more often when teams are co-located or located in proximity to each other.

On our power system software project, being close made it easy to ask for clarifications of the requirements, and saved us time and reduced the defects-count increase. As a result of pair programming and collective efforts of a multi-disciplinary team, the defects life-time had reduced visibly compared with the period before Agile methodologies adoption. This fact was supported by studying the life time of a series of defects during a predefined time frame. The graph in Figure 1 shows how applying Agile methodologies improved the average defect time span.

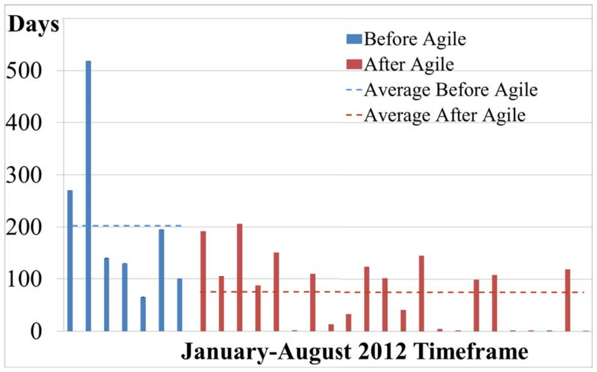

Figure 1. Defects life-time curve

Sometimes when multidisciplinary teams are located together, the environment may become too noisy. Research studies on disciplinary team colocation showed that sometimes productivity decreases instead of increasing [10]. In an open space environment the level of noise can reach 80 decibels, while 65 decibels represents the threshold at which you heart rate reaches the heart-attack level [11]. In the effort of eliminating noise, the team members may inadvertently shut down the channels of communication. If the level of noise is too high in the office, office-mates will use headphones to filter the noise, but that will filter the technical discussions and disable the knowledge transfer. Scientific studies showed that people seeking help from others in open environments are becoming more productive, while the people offering help are having a decrease in productivity [10]. The interruption of your work to help others requires reacquaintance with the details of your task every time you come back to it. Therefore, the best approach is to set time periods when you cannot be disturbed and periods when you are willing to help others.

3.2. Technology to Help Work

Every company has its own preferred tool for editing, debugging, tracking defects and requirements, monitoring of the product life-cycles, revision control systems for code and releases, for updating documentation and so on. One thing it is clear: it is hard to have exposure to all the available products in the market. The software developer of a product should be able to master the use of the company choices in tools, optimize the tedious and time-consuming tasks by making use of the available tools.

Working on a team (distributed or not, collocated or not) requires interaction with the other teammates. The use of instant messages, conference calls, screen sharing and google hangouts represent the routine part in a developer day. Interactions occur all the time during the work day and sometimes there is the danger of losing track of temporal knowledge. Answer to a question like “why has this approach been chosen?” represents temporal knowledge that may get lost in the development process. In the Agile methodologies, the team writes notes, keeps a history of the sprints and backlogs, defects and translates everything in work items. Discussing and taking decisions on the problems with the whole team improve the knowledge exchange and keep everyone equally involved in the development process.

Many companies agree that cloud computing is the direction towards which things are going [12]. The integration of the cloud-based services is in its early days. Based on the needs of the organization, one can use the commercial public cloud services or build one’s own private clouds, or do both. Cloud computing providers offer their services according to several fundamental models (i.e., SaaS (Software as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service)). Using cloud computing will allow the companies to maximize the effectiveness of the shared resources and it can be viewed as a way of increasing productivity.

Even in the scientific software community the cloud computing and parallel programming are gaining more ground, being adopted as ways to improve quality of software and productivity. The increased complexity of the modeled problems require high performance computing capabilities, dedicated platforms and sometime the possibility to integrate hardware equipment and software parts.

4. Communication

Communication is the key to avoid "muddy requirements" [13], by allowing transformation of tacit knowledge, ambiguous assumptions and beliefs into precise specification formulations. The "mud" comes from the different knowledge of the application subject experts and the software experts: communication is the only solution. Engineers must create a functional specification from the marketing concept proposal to document specific features for implementation. The original concept is likely written in business-readable language, and its translation into technical functional specifications can introduce ambiguous and inconsistent requirements. It is important for the functional specification document to be customer reviewed and validated in order to enforce its technical integrity.

Migration from one phase of the project to the the second one may help individual knowledge to be transferred to the whole group. In many instances, in our scientific development project, the power-systems engineers discussed the problem to be modeled with the team, helping to choose the optimal approach. This type of knowledge sharing can be viewed as a way of saving each other's time, if one accounts for the quality of the code produced. In our studied case, a productivity increase by employing pair programing was observed; the continuing peer-review of the software generated more flexible, cleaner and high quality code. All the team members remained in contact with each other and were aware of the project needs and directions. The team gained experience in risk assessments and was able to prevent failures by thinking ahead in the development process.

Communication should occur at all levels in a team. The internal team likely has dedicated communication channels in place, but communication with customers is a bit more difficult. If there is no direct channel of communication between developers and customers, and all the communication is accomplished through the business representatives, many requirements/features can be miscommunicated or misinterpreted.

For illustration purposes, let us include an example that describes the unforeseen troubles of the problem of power factor in a utility. Broadly speaking, the problem was to keep the power factor near unity at a substation. The solution implemented gave results as presented in Figure 2. The diagram shows the result of an automatic control system, operating over several days. It is very clear that when the control is activated, the power factor is maintained very close to unity.

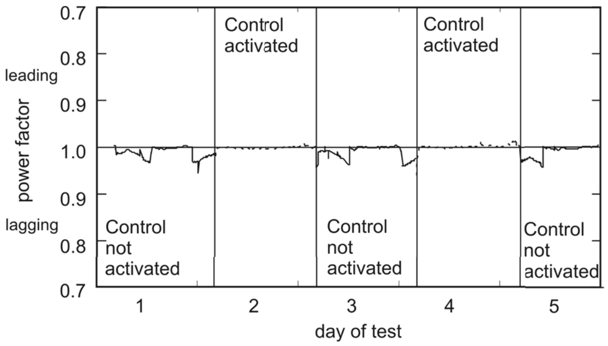

Figure 2. Power factor at bus for 5-day period

The power factor on the substation was being well-controlled. However, closer investigation revealed a problem. The power factor on the secondary bus of the substation was indeed quite well controlled, but on the various lines that were fed by the bus, it was far from well managed. An example of feeder power factor corresponding to the time shown in Figure 2 is given in Figure 3. It can be seen that the feeder has very large deviations from unity power factor when the control is active, not very different from when it is not activated. While the control system is clearly capable of solving the problem, the problem was not well-defined, and the *wrong problem* was solved.

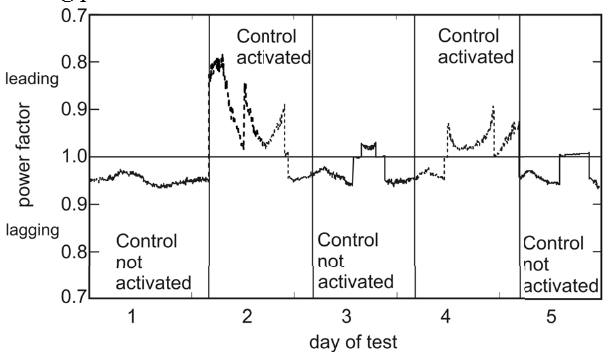

Figure 3. Power factor on line for same 5-day period as Figure 2

There is a trend today to involve the software developer in customer interaction. They can interact with them directly, and get direct user feedback. Feedback is brought back into the product development life-cycle, and that translates into a decreased risk in customer product acceptance. Having a shorter feedback loop can also be reflected in productivity.

Software engineers are more visible in media, social networks and at conferences, providing the audience with direct information on products and the ways they were designed. Usually, software products are not the result of an individual software developer, especially in the commercial companies. Even if each developer is only responsible for a specific part of a large software product, they can and should be part of the team promoting the result. It is hard to say whether or not this would positively influence the product distribution. For example, would a chat with one of the software developers affect the decision of a car buyer, even if the car depended on millions of lines of code? Nevertheless, some developers are becoming more involved in assisting their business in making its value known in the world.

5. Conclusions

The future for the software developer looks bright. Demand among government, scientific communities and business entities to leverage new data sources for business advantage is growing. Digital control is replacing analog. User interfaces are everywhere. Companies can tailor their customer offerings and develop tighter connections with them. These industry trends are creating a demand for greater numbers of software engineers, but they are also changing the role of software engineers. The software engineer of tomorrow will need education and experience beyond that of the software engineer of yesterday.

7. References

[1] Viktor T. Toth, Slava G. Turyshev, “Finding the Source of the Pioneer Anomaly”, IEEE Spectrum, November 2012

[2] John Ousterhout, “Scripting: Higher Level Programming for the 21st Century”, IEEE COMPUTER, 1998

[3] John W. Tukey, “The Technical Tools of Statistics”, http://cm.bell-labs.com/cm/ms/departments/sia/tukey/

[4] Harold Kirkham and Robin Dumas, “The Right Graph: a Manual for Engineers and Scientists”, Wiley and Sons, New York, 2009.

[5] Cristina Marinovici, Harold Kirkham, “The many faces of a software engineer in a research community”, Pacific Northwest Software Quality Conference, 2013

[6] Cristina Marinovici, Harold Kirkham, Kevin Glass, and Leif Carlsen, “Engineering Quality while Embracing Change: Lessons Learned”, 46th Hawaii International Conference on System Sciences (HICSS), 2013

[7] Edward de Bono, “Lateral Thinking: Creativity Step by Step”, Harper & Row Publisher, New York 1973

[8] Robin Whittemore, Susan K. Chase, Carol Lynn Mandle, “Validity in Qualitative Research”, Qual Health Res July 2001 11: 522-537,

[9] Scott Sehlhorst, “To Estimate or Not to Estimate - That Is the Question”, 2013, http://www.techwell.com/2013/05/estimate-or-not-estimate-question

[10] Beth Romanik, “Embrace the Cubicle: Open-Plan Offices Make You Less Productive”, 2013, http://www.techwell.com/2013/05/embrace-cubicle-open-plan-offices-make-you-less-productive

[11] Decibel Levels of Common Sounds, http://home.earthlink.net/~dnitzer/4HaasEaton/Decibel.html

[12] Eric Knott, Galen Gruman, “What cloud computing really means“, http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031?page=0,0

[13] Ivy F. Hooks, Kristin. A. Farry, "Customer Centered Products: Creating Successful Products Through Smart Requirements Management.", AMACOM - American Management Association, New York, 2000