A Gentle Introduction To Robotics Software Engineering

Research · April 2015

DOI: 10.13140/RG.2.1.4329.9361

Birol Aygün
Yeditepe University
7 PUBLICATIONS **16** CITATIONS

A Gentle Introduction to Robotics Software Engineering Education

Asst. Prof. Dr. Birol Ö. AYGÜN
School of Engineering, Dept. of Computer Engineering
Yeditepe University, Istanbul, Turkey (baygun@cse.yeditepe.edu.tr)

Abstract

As robotics becomes increasingly important in technology and therefore in our contemporary society, engineering education needs to increase its offerings related to this new discipline. Since robotics involves topics from several branches of engineering and science, students in this branch need a synthesis of material and techniques from different areas. Our approach is to look at the project lifecycle in robotics software engineering education in the light of experience in existing branches of engineering and to focus on how a synergy can be realized to achieve success in educating students in this new discipline.

Keywords: *software engineering, robot, robotics, software engineering education, mechatronics*

1. Introduction

The general principles for software engineering apply to robotic software engineering as well. We can break down the subject matter into two areas: academic aspects and practical aspects. A subject typically examined under academic aspects is the difference between "software" engineering, in which the products are software, and "hardware" engineering, in which the products are hardware. Differences between these two types of products before, during and after development are emphasized. Robotic software engineering contains a large intersection with robotic hardware engineering due to the high level of dependencies and interactions among robotic components of these two types. The reason for the term "Gentle" in the title is due to a concern that if this multi-disciplinary multi-faceted subject is not introduced in a very well-organized way, the development process can get extremely complicated and the developers may get bogged down easily. Hence to preserve and enhance the enthusiasm the students start with, a well-organized "gentle" presentation by the instructor and well-organized teamwork is vital. For more details on the so-called Team Software Process ^{TM}, see [11]. We also include in the Appendices excerpts from the ACM Curriculum for Robotics courses and IEEE Robotics & Automation publication on Society & Education.

We now discuss the content of practical aspects of these two domains. While there are several approaches to the software development cycle (SDLC) such as Agile and Kanban [19]. In this paper we shall use the classical software Waterfall development cycle [20] which generally consists of the following phases in software product development:

i) **Stakeholder and General Requirement Identification**
In this phase, the students learn about the importance of the so-called "stakeholders" in the project, i.e. groups of individuals or organizations who have a risk and potential benefit from the project. Typically these are the customer and his organization, and even the society in general, the individual users in that organization, the software development company and the individual developers. An output of this phase is a statement of the general requirement(s) for this project.

ii) **Domain Analysis**
Having identified the general requirement for the project, a more detailed analysis of the domain to identify the "actors", their "roles" in the domain and the results of their activities.

iii) **Requirements Analysis**
Detailed analysis of the requirements of the project as accepted by the stakeholders, generally included in a Software Requirements Specification document.

iv) **General Design**

An initial overview of the design for the project based on the information gained from the above phases, typically including overall UML diagrams.

v) **Detailed design**

A more detailed elaboration of the General Design, which can be later used as a guide in the implementation.

vi) **Implementation, testing, evaluation**

The actual code development phase, with an a work plan including a schedule, group responsibilities, an integration plan, a testing plan, evaluation and perhaps additional design and development until the requirements are met to the satisfaction of development management.

vii) **Maintenance and re-engineering**

These phases occur after the product has been installed and put into use. Any problems discovered during use are solved and applied to the product. After a certain amount of usage, if enough new needs are discovered, the product may be re-enginered to resolve the problems and make improvements.

We should note that documentation for each of the above phases should be completed at the end of the phase or shortly thereafter while the ideas and related information are still fresh in developers' minds. These steps may not be followed strictly in this sequence, especially where there multiple groups on the same project. Further, almost always there will be a need to revisit an earlier phase to make some changes. Each such revisit to an earlier phase may require repeated revisits to even earlier phases. There are different formulations of project development phases similar to the one in [21], but in general, most formulations can be accomodated in this structure, called a “Waterfall with Feedback” model, a modification of the classic Waterfall Model. A more detailed treatment of this subject is outside our topic since the project development flow is similar for robotic software engineering. The below figure illustrates this approach, with the addition of feedback from each phase to the prior phase to enable changes in the earlier phase:

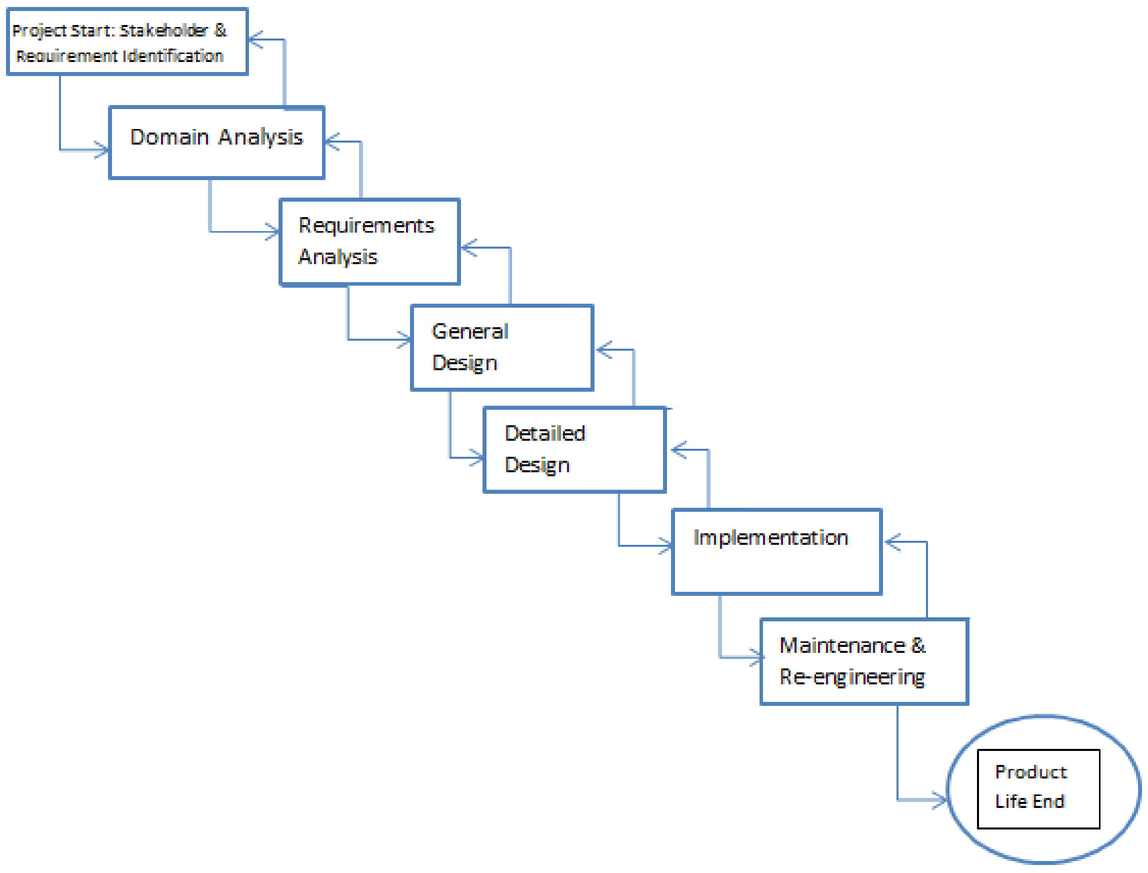

Figure 1. General Product Development Lifecycle

Some unique aspects of robotic software design and development are as follows:

- Many disciplines are involved (mechanical, electrical, digital electronics, software engineering, human factors, possibly others related to the specific domain)
- Complexity and varieties of robot architecture, functionalities and hardware components
- Each application involves some combination of fields and components mentioned above
- Robotic parts are expensive and evolve quickly

Hence the need for simulation of robotic systems by software has evolved. An overview of some robotics simulation tools is included in [12].

Below is a diagram of software lifecycle after initial development:

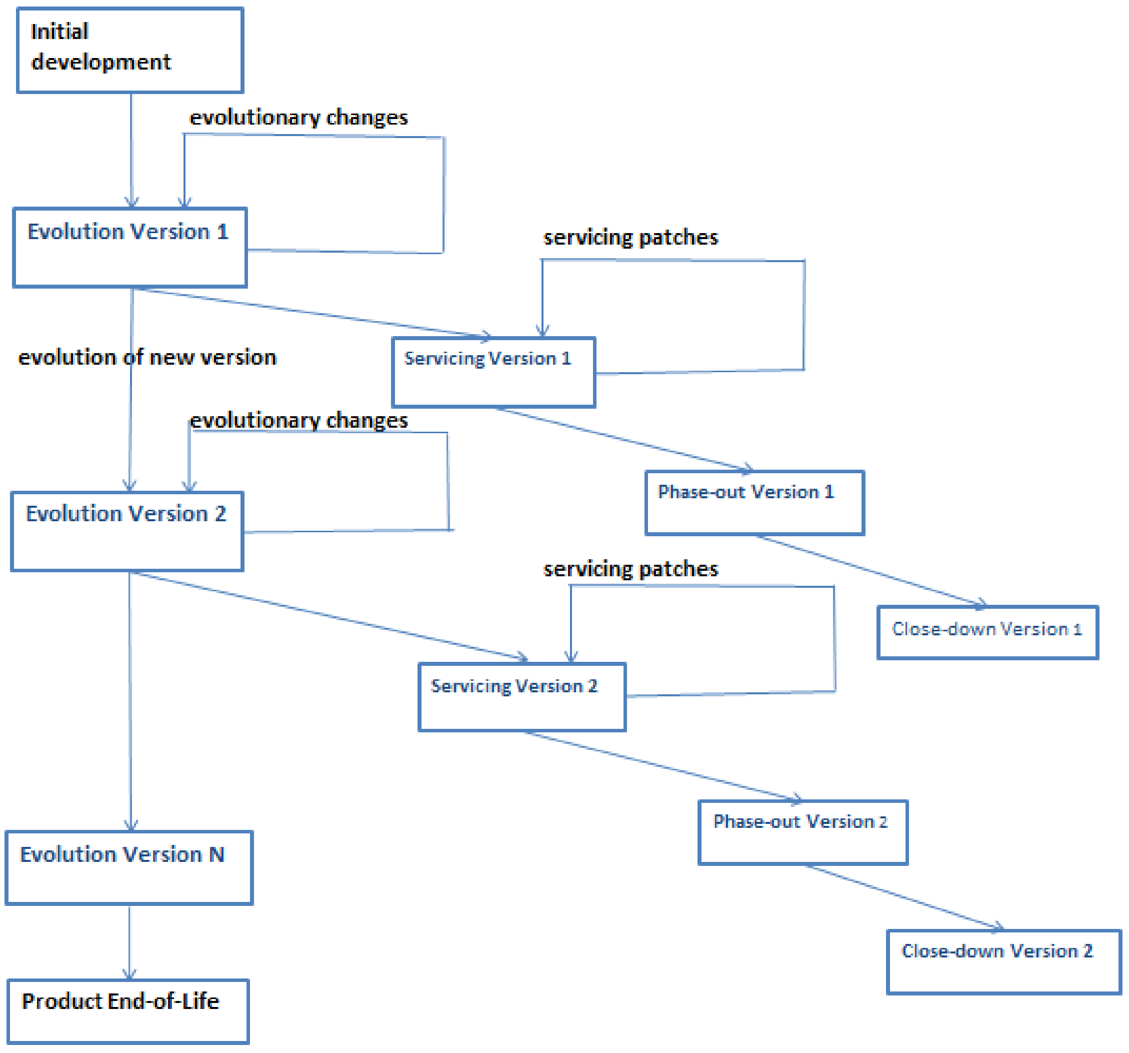

Figure 2. The staged model of software life cycle after release, adapted from [4]

2. Some approaches to robotic software development and robotic analysis

Based on the availability of some simulation models of a robotic components, a number of different approaches to robotic software development has evolved:

- Total software simulation
- Partial simulation and software-augmented hardware
- Software driving both simulator and hardware robot (e.g. assembling parts in a factory)
- No simulation during normal usage

Due to the hybrid nature of robotic architecture including both hardware and software architectures, simulation tool development can be complex. We briefly discuss these alternative approaches below.

2.1 Pure software simulation

This approach can be taken with traditional simulation tools and/or programming languages such as C, C++, Java and Python augmented with a suitable computation package such as Matlab [15], with or without Simulink [16]**.** In fact, software languages and platforms supporting multi-threading or distributed processing can be used effectively to create the desired simulations**.** Some simulation techniques are based on pure text output which can be analyzed by a post-processor to make inquiries into the events that took place during the simulation. However, often graphic displays are an important part of a simulation package. If 2D or 3D graphics are required, suitable graphics tools supported by a physics package may be used. For more details, see [12] for examples.

2.2 Software-augmented hardware

During development or adaptation of a robot hardware system for a given application, not all the hardware parts may be available. Therefore software simulating the missing parts can be used to represent, and in fact to experiment with, different designs for the missing parts. Often this kind of software is designed to be able to drive the missing hardware part as well when and if it is later installed.

2.3 Software driving both simulator and hardware robot

Extending the above idea, when designing a simulator for a given robot architecture, it is often desirable for the same software to drive both the simulator and the actual robot. This way faithfulness of the software model to reality can be tested and the simulator can be used by multiple designers simultaneously to reduce expenditure on the robot. [12]

2.4 Robot Usage without Simulator

In this stage the robot is being used without a simulator either during development or actual intended use. Data may still be collected for subsequent analysis and evaluation. For example, data about the behaviour of various components such as sensors, actuators, movements of the components and the like may be collected analyzed for correctness and fluctuations over time.

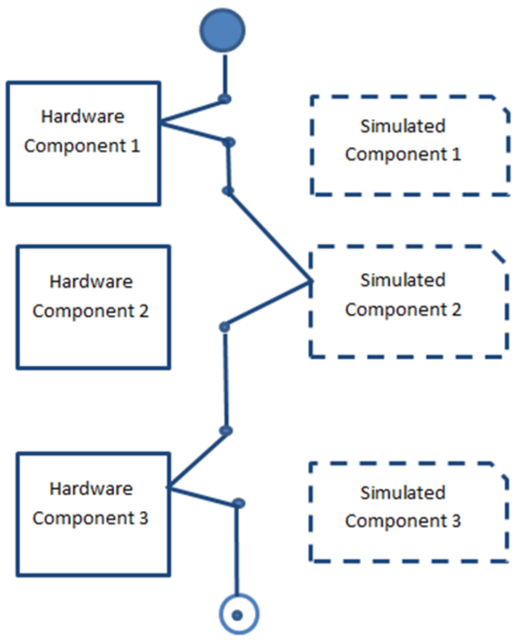

Figure 3. Structure of a robot containing hardware components and simulated components

In the above figure Hardware Component 2 has been switched off and Simulated Component 2 has been switched on in its place.

3. Training of Software

To increase the accuracy of the simulation software, it may be necessary to collect data from the actual hardware and train the software based on data collected from itself by comparing it with data from the hardware by using methods of “supervised training” from the field of data mining and artificial intelligence. This may be used to improve the vision, object recognition, path selection speech recognition,and similar capabilities of the robot. If no such data for the hardware is available, methods of “unsupervised training” need to be used, which are usually less accurate and have lower levels of confidence [2].

4. Mapping of Software Structure to Architecture & Design Patterns

Another concept that can be brought to bear in software development is the concept of architectural and design patterns [18] . Since a given family of robots will have similar sets of reasoning and acting abilities, these patterns can be created and stored in a library, or better yet in an ontology [13], to simplify software development. Studies have been done to facilitate mobile robot software knowledge reuse by capturing conceptual models. These studies may be included in a robotics software engineering curriculum to illustrate techniques for reusing robot software knowledge. [3]

5. Seperation of Concerns

One of the problems in the application of system enginering techniques to robot design and development has been identified as “seperation of concerns” [1], an issue familiar to software engineers. The issue has been raised as a contributor to the current lack of expected levels of achievements of “robot performance in real world environments”. We do not replicate the issue here but wish to bring up its consideration in a robotics software engineering course curriculum. Interested readers can access the details in the referenced publication.

6. Managing Fuzz

A concept that robotic software developers come in contact with is that of so-called “fuzz”. In robotics “fuzz” refers to complex uncertainties in the state of the robot or robot group that come about due to minute inaccuracies the responses of hardware and software in the actual robot or simulated robot. These inaccuracies can occur in the performance and measurements of real or simulated mechanical movements, forces and energies involved, part specifications, materials, the environment, and electronic/electrical fluctiuations in the parts and measurements. When such inaccuracies are compounded, they may be difficult to isolate and keep within allowable ranges.

Another complex issue here is the optimization of deviations due to fuzz and sensitivity of measurements. This is likely to be one of the most time-consuming parts of the project due to the uncertainties in the nature of data collected, hence the word “fuzz”. Thus, careful experiment design and collection of relevant data of sufficient accuracy is of utmost importance. This topic can be briefly presented in the initial course with more complex topics introduced in advanced, multi-discipline courses. For a broad overview of the subject and a good reference list see [10].

7. Data management, experiment repeatability, performance analysis and evaluation

As is apparent from the preceding, effective data collection is very important to achieve optimum behaviour from the robot. This means using proper tools and techniques needed to obtain correct results from the experiments. These may include simple sequential files, indexed files, relational databases, probabilistic data in ontologies[13] and other formats. Sufficient repetitions of experiments should be made until the above analyses produce results with required levels of statistical confidence for the particular application. [2]. For even better analysis, results obtained from the above data should be compared with new data obtained independently.

8. User Interfaces

Several types of user interfaces are needed, such as interfaces to:

(i) Develop the software, including interfaces to available, existing software or the framework being used. Typically a library to house the collected data in the form an ontology including heuristic data analysis and modelling capabilities should be available

(ii) Run the software/hardware robot system, including interfaces for data collection, perhaps in online and offline status due to length of runs

(iii) Analyze data and generate needed reports,

(iv) Make needed software modifications through interface type (i) above.

Requirements and use-case design analyses of the interfaces should be made early and used to guide their development [14].

9. Documentation and Reengineering

An important part of engineering education is to create technically sound and easily understandable documentation for one's work. This is especially true for projects. The instructor should plan the content and format for standard student projects. The conformance of the projects to this format should be a part of the project evaluation. Reengineering often appears as extensions or changes to the content of the project as delivered. Hence, if the project can be re-engineered easily and succesfully, it is a sign that the project is well-structured and documented. For robotics software, documentation should address subjects in different fields of engineering, overall system views of robots and experimental data related to the robot.

10. Teamwork

Teamwork in undergraduate projects is very important to give students experience in working with colleagues to obtain the desired results. So the work plan for the project should be designed to give every member of the group a significant role in all phases of the project mentioned in the beginning of the paper. Sometimes this is hard to achieve due to schedule conflicts and differences in abilities and motivation levels of the team members. A robotics project is often a multi-disciplinary project, including subjects from several disciplines, as those disciplines are currently defined. Therefore, one approach is, where possible, team members should be given tasks closest to their disciplines. Another approach, which can be called the "mix-and-match" approach, may be for members of different backgrounds to work together on a given part of the project to increase their exposure to problems outside their specialties.

However, in all cases, the project and the documentation as mentioned above or as prescribed by the instructor should be completed to the instructor's satisfaction.

11. Sample Curricula Recommendations

11.1 Excerpt from ACM Computer Science Curricula 2013

"IS/Robotics[Elective]

Topics:

- Overview: problems and progresso State-of-the-art robot systems, including their sensors and an overview of their sensor processing
- Robot control architectures, e.g., deliberative vs. reactive control and Braitenberg vehicles
- World modeling and world models
- Inherent uncertainty in sensing and in control
- Configuration space and environmental maps
- Interpreting uncertain sensor data
- Localizing and mapping
- Navigation and control
- Motion planning
- Multiple-robot coordination

Learning Outcomes:

1. List capabilities and limitations of today's state-of-the-art robot systems, including their sensors and the crucial sensor processing that informs those systems. [Familiarity]
2. Integrate sensors, actuators, and software into a robot designed to undertake some task. [Usage]
3. Program a robot to accomplish simple tasks using deliberative, reactive, and/or hybrid control architectures. [Usage]
4. Implement fundamental motion planning algorithms within a robot configuration space. [Usage]
5. Characterize the uncertainties associated with common robot sensors and actuators; articulate strategies for mitigating these uncertainties. [Familiarity]
6. List the differences among robots' representations of their external environment, including their strengths and shortcomings. [Familiarity]
7. Compare and contrast at least three strategies for robot navigation within known and/or unknown environments, including their strengths and shortcomings. [Assessment]
8. Describe at least one approach for coordinating the actions and sensing of several robots to accomplish a single task. [Familiarity]"

11.2 An excerpt from IEEE Robotics & Automation publication on Society & Education

"...Some introductory courses stress parallel processing from the outset (with traditional single threaded execution models being considered a special case of the more general parallel paradigm). While we believe this is an interesting model to consider in the long-term, we anticipate that introductory courses will still be dominated by the "single thread of execution" model (perhaps with the inclusion of GUI-based or robotic event-driven programming) for the foreseeable future. As more successful pedagogical approaches are developed to make parallel processing accessible to novice programmers, and paradigms for parallel programming become more commonplace, we expect to see more elements of parallel programming appearing in introductory courses."

12. Conclusions

Despite the current nascent state of robotic software engineering, software engineering education should include introductory material in this area, with possibly more advanced material in specific courses on the subject. One of the problems will be to keep these courses up-to-date with the rapid introduction of new robot technologies and related engineering methodologies. While we have above delved into many aspects of robotics software engineering, not all aspects can be addressed in an introductory course. The burden, as usual, in on the instructor's shoulders to create a cohesive syllabus from these, and possibly other areas. In fact, a series of 2 or 3 courses covering robotics software can be developed by extending the material above.

The trend toward on-line education programs will help in keeping course materials, including student projects, updated. This can be further helped by use of simulators of robots and robot teams whose software can be updated easily. In fact, student projects can be used to develop updates to the simulators. Laboratories with highly parallel architectures can be used to simulate robot groups and related control technologies while working on actual robots. Industries will want to subsidize research in their areas of production and even employee training. This will give more incentives to universities to create and expand research on robotic systems.

While it is hard to estimate the rate of progress in robotic software technology, we can expect that as we understand applications of this technology better, the success rate will be higher than the current rate, similar to applications of many new technologies in the past in such areas as in automotive, electronics and other engineering disciplines.

References

[1] C. Schlegel, A. Steck and A. "Robotic Software Systems: From Code-Driven to Model-Driven Software Development", Robotic Systems - Applications, Control and Programming, Dr. Ashish Dutta (Ed.), ISBN: 978-953-307-941-7, InTech, 2012.

[2] I. H.Witten, F. Eibe,"Data Mining: Practical Machine Learning Tools and Techniques", Second Edition (Morgan Kaufmann Series in Data Management Systems) Morgan Kaufmann Publishers Inc. San Francisco, CA, USA ISBN:0120884070, pp 83-283, 2005.

[3] D. Jawawi, S. Deris, and R. Mamat, “Software Reuse for Mobile Robot Applications Through Analysis Patterns”, The International Arab Journal of Information Technology, ISSN:2309-4524 (online) http://ccis2k.org/iajit/?option=com_content&task=view&id=163, last accessed 20/03/2015

[4] D. Brugali, P. Scandurra, A. Gargantini et al, “Design Principles, implementation guidelines, evaluation criteria, for system openness, and flexibility and use case implementations”, BRICS Best Pratice in Robotics, Universita degli Studi di Bergamo, September, 2010.

[5] S.Becker, C.Brenner, C.Brink, S.Dziwok et al, “MechatronicUML – Process, Syntax and Semantics” Technical Report tr-ri-12- 326t , Software Engineering Group, Heinz Nixdorf Institute, Paderborn, Germany August 2012

[6] L.Kunze, T.Roehm, and M.Beetz, “Towards Semantic Robot Description Languages” 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, May 9-13, 2011

[7] E. Gamma, R.Helm, R.Johnson, J.Vlissides, “Design Patterns – Elements of Reusable Object—Oriented Software”, Addison-Wesley Longman, Inc.; ISBN 0201633612, 1995

[8] Jager, R & Filev, D.P. “Essentials of Fuzzy Modeling and Control”, John Wiley-Interscience Publication, New York, 1994.
.
[9] Boreinstein, J. & Koren, Y., “Real-time obstacle avoidance for fast mobile robots”, IEEE Trans. on Systems, Man., and Cybernetics, Vol. 19, No. 5, Sept/Oct. pp. 1179-1187, 1989

[10] Jamshidi, M., Vadiee, N. & Ross, T. J., “Fuzzy Logic and Control. Software and Hardware Applications”, PTR, Prentice Hall, New Jersey, USA, 1993

[11] Humphrey, W. Introduction to the Team Software Development Process, Addison-Wesley Professional, 1999 Addison-Wesley, SEI Series in Software Engineering, 1999

[12] Staranowicz, A., Lucatini, M.G, “.A Survey and Comparison of Commercial and Open-Source Robotic Simulator ”, Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments Article No. 56 ACM New York, NY, USA ©2011

[13] Schlenoff, C. et al “An IEEE standard ontology for robotics and automation”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1337-1342, 2012

[14] Lethbridge, L. & Laganiere, R., “Object-Oriented Software Engineering”, 2. ed., McGrawHill 0-07-710908-2

[15] http://www.mathworks.com/products/matlab/ last accessed 26 March 2015

[16] http://www.mathworks.com/products/simulink/ last accessed 26 March 2015

[17] Jawawi, D. et al, “Software Reuse for Mobile Robot Applications Through Analysis Patterns”, The International Arab Journal of Information Technology, Vol. 4, No. 3, July 2007

[18] Gamma, E. et al, “Design Patterns: Elements of Reusable Object-Oriented Software” Addison-Wesley. ISBN 0-201-63361-2, 1994

[19] Ambler, S. & Associates, “The Agile System Development Life Cycle”, http://www.ambysoft.com/essays/agileLifecycle.html last accessed 23 April 2015

[20] http://en.wikipedia.org/wiki/Waterfall_model last accessed 23 April 2015

[21] http://www.informatik.uni-bremen.de/gdpa/def/def_w/WATERFALL.htm last accessed 23 April 2015

Authors: Dr. Birol Aygün holds B.S.M.E degree from Newark College of Engineering (now New Jersey Institute of Technology), M.S. degree in Engineering Mathematics from Columbia University School of Engineering and Applied Science and Ph.D. degree in Computer Science from Carnegie Mellon Univesity. He has worked for IBM for seventeen years in the areas of advanced applications and evaluation of the reliability and performance of large systems software. He previously taught at City College of New York and Vassar College in New York. Currently he is an Asst. Professor at the School of Engineering, Dept. of Computer Engineering, Yeditepe University in Istanbul, Turkey.