An Effective Development Environment Setup for System and Application Software

Aleksandar Bulajic
Metropolitan University, Belgrade, Serbia
aleksandar.bulajic.1145@fit.edu.rs

Samuel Sambasivam
Azusa Pacific University, Azusa, CA, USA
ssambasivam@apu.edu

Radoslav Stojic
Metropolitan University, Belgrade, Serbia
Radoslav.Stojici@fit.edu.rs

Abstract

The agile development method does not represent a single approach, but rather defines a number of recommendations used in Extreme Programming (XP), SCRUM, Test Driven Development (TDD), and other methodologies that implement an agile software development system. Besides studies of the Integrated Development Environment (IDE) and automated test tools promoted by Extreme Programming and Test Driven Development, it is not easy to find information in the current literature about an effective software development environment where different tools are combined to automate software development tasks, and replace error-prone and time-consuming manual work. This paper is an attempt to fill this gap, and to draw attention to the fact that an effective development environment requires adequate tools. After presenting different tools that can be used for application lifecycle management, change management, collaboration, development, test, build and deployment automation, and continuing integration, this paper offers a model that recommends a set of tools that can work together to provide an effective development environment.

Keywords: Agile, Effective development environment, Environment setup, Continues Integration, Automated deployment

Introduction

Software development is the construction of system or application software that can be deployed on computers to execute repetitive tasks; more or less this software automates manual tasks. Software development is very often understood as programming, or writing a source code by using a particular computer language, but it can also include all other tasks from an initial idea to the implementation of the same idea. Im-

plementation in this case means specification of all details or algorithms, design, programming, testing, deployment, and maintenance.

While most of the current literature is dedicated to the description of methodology and processes (Sommerville, 2001; Larman, 2005; Leffingwell & Widrig, 2000; Beck, 2002; Beck et al., 2001; Beck et al., 2001a), a very important part, the implementation platform, is often ignored. The reason is probably based on the understanding that the methodology should be independent from implementation details.

The description of an implementation platform is ignored, and it is assumed that a method implementation requires a computer. This assumption is, of course, correct; because in order to execute the software we need a processor and hardware.

This paper is an attempt to fill this gap information, and to draw attention to the concept that an effective development environment requires adequate tools. To be productive and effective, any kind of production requires tools, no matter where—in civil engineering, construction, mechanical engineering, or software engineering. Today it is unimaginable to even think about producing machine parts manually. This is a job for highly specialized machines connected to the computers, with their operations managed by Computer Aided Design (CAD) systems. Some will argue that software development is different, and of course it is, as any production type is different; however, the use of tools is a common pattern, and replacing manual work improves productivity, reduces the number of errors, and, generally speaking, significantly improves the average product quality.

While it can be argued that these tools and the development environment setup are important for any software development method, we believe that an effective development environment is more vital in the agile and distributed development environment. In the case of the agile development approach, waiting a day or two for a new release to be deployed and tested is not an option. Short iteration requires prompt feedback. Also, what is very important today, when outsourcing is present, is prompt and timely information-sharing and storing, even in small development teams. Although communications through e-mail messages, chats, and over video conferences can help to share information and quickly clarify issues, they do not guarantee that information, as well as any conclusions or actions, will be stored in a repository that enables fast and easy searching, filtering, or retrieving. An effective collaboration environment needs tools that can inform and warn the stakeholders about any kinds of changes.

Setting up an effective development environment in advance is very rare. Development environment setup, as well as using of proper tools, is mostly left to the particular developer. Although many IT companies are aware of the need to use an effective development environment and standard set of tools, very seldom does this awareness assist in the choosing of an Integrated Development Environment (IDE) framework and target database software. Aside from creating an ineffective use of tools, this approach can also create a lot of issues in the future. An example issue is the replacement of skilled developers with those that are not familiar with the tools, or tool installation and configuration.

An effective development environment can significantly speed up the software development process and improve final product quality. Tools that were once propriety products are now affordable for most companies. Most of the tools are open-source products and can be used free of charge for development of any kind of software. Software development tools, like Integrated Development Environment (IDE) frameworks, automated testing tools, continuous integration tools, communication tools, and other tools can significantly improve effectiveness and productivity. Many of these tools are able to generate code or help functions. Some examples are the project wizards that create specific project templates, the automatic generation of Web services classes and descriptors, or the creation of a database from a class description. These functions are a

standard functionality in the most popular and competitive Integrated Development Environments (IDE), such as Eclipse and Microsoft Visual Studio.

This paper is based on practical experience collected from a huge number of projects, and on the research and analysis of the currently available tools that can be used to create highly integrated and collaborative software development environment, that are also free or affordable for all types of project budgets and sizes.

Literature Survey

Effective development environments and tools are not a well-supported topic in the Software Engineering literature. The integration between different tools is mentioned mostly in practitioners' blogs by those searching for solutions and sharing experiences of ideas already implemented as workaround solutions. Even in these blogs, an effective software development environment is not mentioned as a main subject. Discussions are mostly regarding particular product installation, configuration and compatibility issues related to newer versions, or compatibility issues related to collaborations with other specific products.

Sommerville (2001) briefly described Computer Aided Software Engineering (CASE) tools for change management, and emphasized the importance of release management, build, and versioning control. Besides mentioning a few distinct CASE tools, such as Lifespan and Domain Software Engineering Environment (DSEE), the author dedicated a chapter to Change Management, but did not include more information regarding to development environment.

Even such great practitioners and agile methodology founders and gurus such as Martin Fowler, Kent Beck, and Jeff Sutherland, do not discuss development environment and tools besides unit testing tools, code refactoring, and issues related to the using of particular programming languages. (Fowler, 2006; Beck, 2002; Beck, et al. 2001; Beck ,et al. 2001a; Sutherland, 2011)

Larman (2005), another astute guru, is dedicated to an effective project management, and describes Use Cases and the implementation of UML diagrams. Larman recommends black-box use cases as "the most common and recommended kind; they do not describe the internal workings of the system, its components, or design. Rather, the system is described as having ***responsibilities***, which is a common unifying metaphorical theme in object-oriented thinking—software elements have responsibilities and collaborate with other elements that have responsibilities. By defining system responsibilities with black-box use cases, one can specify ***what*** the system must do (the behavior or functional requirements) without deciding ***how*** it will do it (the design)." (Larman 2005) Larman also mentioned the Unified Process and the more complex implementation of Unified Process (UP), called the Rational Unified Process (RUP).

Leffingwell and Widrig (2000) provide an extensive description of the RUP and the management of software requirements, and assume that the entire process is managed by an RUP tool, based on the needs, features, use cases and requirements, and test case hierarchy verification. However, with the exception of a detailed process description and high architecture overview, there is no mention of development tools, although it can be argued that the IBM RUP itself is a development tool.

Forrester Research, Inc. (Grant et al. 2012) provides an excellent analysis of the Application Lifecycle Management tools, but does not provide information on how these tools can be integrated to an effective development environment. Forrester Research, Inc. (Grant et al. 2012) noticed that "IBM provided multiple solutions in some of these areas, leading to confusion about what exactly each of these tools was supporting and how they all worked together." IBM Rational tools are integrated into the IBM development tools; for example, Rational Application Developer (RAD). This development environment enables visual modeling by using UML and accelerates

development. IBM RAD generally supports Java technologies, Web and mobile application development, Java EE, and SOA. These kinds of tools are expensive, very specific, and not well adapted for integration with third-party tools.

Frederick Brooks (Brooks, 1995, p.196) in the "The Mythical Man-month", offered a quick discussion about a programming environment and the necessary tools, and concluded "perhaps the biggest gain yet to be realized in the programming environment is the use of integrated database system to keep track of the myriads of details that must be recalled accurately by the individual programmer and kept current in a group of collaborators on a single system".

If the Internet is searched by using "effective development environment" keywords, the search result will show different subjects that are mostly related to the description of different environments used for development, test, pre-production, and production. Another set of results is related mostly to the particular development process, such as test or development, or to the particular development framework, such as Eclipse, Microsoft Visual Studio, or NetBeans, and tools that are delivered and built-in within each of these particular frameworks.

If a search is executed using "development tool" keywords, the search result will show development tools from major software vendors, such as Apple, Inc.("Developer tools" 2013), Oracle ("Oracle developer tools" 2013), Eclipse Foundation ("Eclipse Java development tools (JDT) Overview" 2013), Microsoft ("Developer resources" 2013), and Android ("Android Developer Tools" 2013), and then a list of the tools and utilities developed by middle-sized and small software companies.

Each of the software vendors describes its own product and offers an integrated development environment that incorporates editor, debugger, compiler, and linker, and provides a number of tools that enable unit testing, syntax highlighting, project wizards, and code generating. Some of these tools are open sources, such as Eclipse and Android tools, but some are licensed products, such as Microsoft Visual Studio, although Microsoft offers a free-of-charge Microsoft Visual Studio Express version that has limited functionality. Some of the tools' functionality can be extended by plug-ins and add-ons, as for example Eclipse.

Eclipse ("Eclipse Java development tools (JDT) Overview" 2013) describes Java development tools (JDT) such as annotation processing support, Java builder, Java Model API for Java element tree that defines a Java-centric view of project, code assists, JDT debug, editor, and JDT UI Java-specific workbench.

Microsoft ("Developer resources" 2013) offers a list of the tools that support software development, and that are integrated in the Microsoft Visual Studio IDE framework. This set of tools is used for code writing and generating, building, and debugging, as well as deployment on different Microsoft platforms.

Although all these tools are important for an integrated development environment, the primary target of these tools is a single developer environment. It is not sufficient for a development team to use only these tools, and in case of multiple development teams spread all around this planet this can be a significant bottleneck that causes serious project delays and reduces product quality.

It is easy to understand why these software vendors do not recommend other software tools than own. That would explicitly admit weakness in one's own product and promote the competitors.

An effective development environment requires that a build can be executed outside of the integrated development environment by executing an automated build procedure. In this case the build is executed as an entire project build, called an integration build, and includes all other code developed by all team members. This approach requires that the build is executed by a script, and

the script shall be executable by any other type of tool, even those from different software vendors.

A continuous integration requires an automated build and deployment process, and these are preconditioned to execute an automated test and provide prompt feedback to the development team. The section labeled "An Effective Software Development Environment Model Example" provides detailed discussion about all of these subjects.

Software Development Model

Today the following Software Development Models (SDM) are widely accepted:

1. Sequential or linear
2. Iterative
3. Evolutionary

All of these methods are implementations of the Systems Development Life Cycle (SDLC) processes defined in the mid-1960s by A. Enthoven and Henry Rowan. (Boggs 2004) Sequential SDM, such as Waterfall and Iterative methods like Agile, are currently the most well-known implementations of the SDLC-based methodologies. The SDLC defines the following processes:

1. Project Planning and Feasibility Study
2. Requirements Definition
3. System Design
4. Development and Implementation
5. Integration and Testing
6. Acceptance and Deployment
7. Maintenance
8. Disposal

Different methodologies describe these SDLC processes using different words or a different number of processes, but all follow the basic SDLC process life cycle pattern, and divide tasks and goals of each phase so the output from each previous phase is the input in the next phase. For example, the Waterfall method can be described by:

1. Analysis
2. Architecture & Design
3. Development
4. Test
5. Deployment & Maintenance

Figure 1, "Waterfall Methodology", illustrate these five phases and expected outputs from each of the phases:

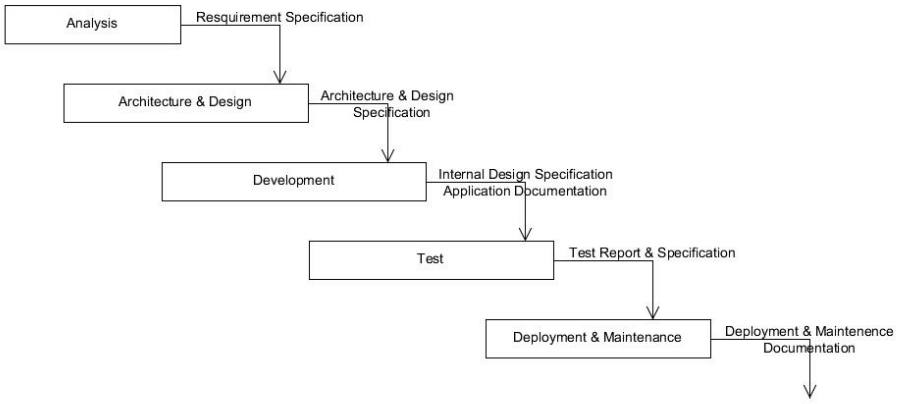

Figure 1 "Waterfall Methodology"

First known record of the Waterfall development method is found in 1956 at the "Symposium on Advanced Programming Methods for Digital Computers: Washington, D.C., June 28, 29, 1956" by Herbert D. Benington (1956) in a presentation of his paper, "Production of Large Computers Programs." Dr. Winston W. Royce in 1970 presented his personal view about managing large software developments in his paper, "Managing the Development of Large Software Systems" at the "Proceedings of IEEE WESCON 26" (Royce 1970). Both papers presented experience collected during many years of working on the development of the large computers programs.

The Waterfall method is most appropriate for a project where requirements are stable and do not change often during the development and implementation phase. However, analysis shows that an average of 25% of the requirements change in the typical project, and the rate of change can increase to 35% to 50% for larger projects. (Benington, 1956; Larman, 2005)

An iterative approach breaks the project into more pieces or phases, where each phase output is functional software that implements a limited set of requirements. The last phase is supposed to deliver fully functional software that implements all requirements and is ready to be deployed in the production environment.

Agile approach is also called incremental and evolutionary. Each phase adds a new value to the existing software and incrementally builds the entire product. The requirements are refined during planning of the next phase, and corrected by a better understanding determined during implementation from client feedback. Iterative and Agile methodologies consist of a number of short Waterfall phases.

Figure 2 "Iterative and Incremental Software Development Method" illustrates the iterative approach and today's most popular agile, iterative, and incremental approach:

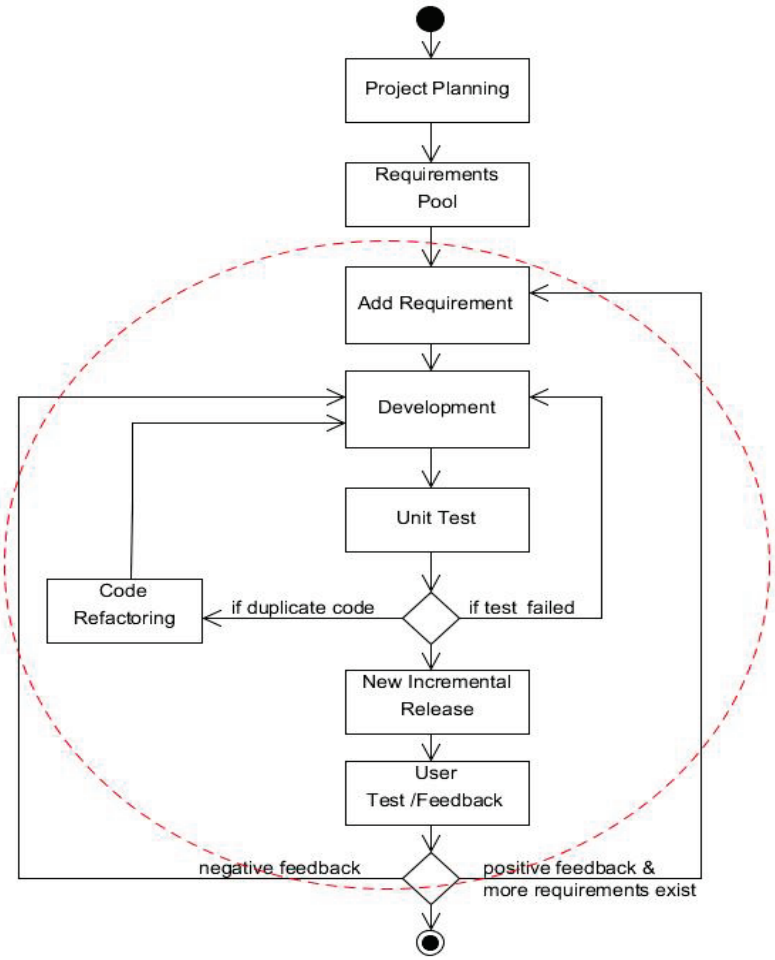

Figure 2 "Iterative and Incremental Software Development Method"

Although many would argue that the agile, incremental, and evolutionary approach is a new concept (Fowler, 2005), 1950s software developers were already aware that the development of the large computer programs were a challenge and suggested different approaches. Herbert D. Benington suggested the application of an evolutionary approach.

"To underscore this point, the biggest mistake we made in producing the SAGE computer program was that we attempted to make too large a jump from the 35,000 instructions we had operating on the much simpler Whirlwind I computer to the more than 100,000 instructions on the much more powerful IBM SAGE computer. If I had it to do over again, I would have built a framework that would have enabled us to handle 250,000 instructions, but I would have transliterated almost directly only the 35,000 instructions we had in hand on this framework. Then I would have worked to test and evolve a system. I estimate that this evolving approach would have reduced our overall software development costs by 50 percent. (Benington, 1956) "

Today, software development is further complicated by outsourcing and united development teams spread all around the globe. Time differences, cultural differences, and the requirement for tight and prompt collaborations create many obstacles that affect effectiveness and quality, and those obstacles need to be resolved quickly.

An effective development environment is even more important in the case of Agile and a distributed development environment. In an Agile development approach, waiting a day or days for a new release to be deployed and tested is not an option. A short iteration requires prompt feedback.

The rest of this research paper describes the tools that can enable effective development, test, and deployment. While it can be argued that these tools and the development environment setup are important for any software development method, we believe that in the case of an agile approach these tools can make important difference between a success and a failure, and significantly improve final product quality.

Although the next section briefly discusses project planning and resource allocation, the primary focus and scope are about the tools that are used during the development phase. In Figure 2 "Iterative and Incremental Software Development Method" the red circle illustrates this scope.

Software Development Project Constraints Definition

Each Software Development project's direct constraints include Time, Budget, and available Resources, as illustrated in Figure 3. The Quality of the project's product depends on each of these three constraints:

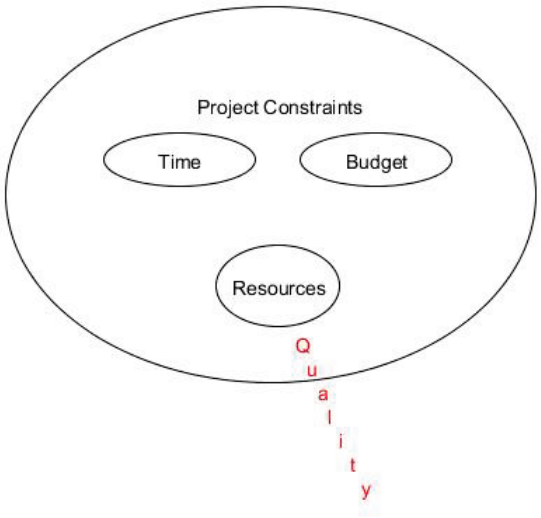

Figure 3 "Project Constraints Time, Budget, and Resources"

The Software Development Project shall meet deadlines and face budget constraints. Resources are directly dependent on Budget size and available time. In cases where the Resources are identified as human resources, their availability can be a limiting factor. If the best resources are occupied, or reserved in advance by another project, then the project can count only on available resources. Budget and time reflects estimated amount of work. Time can be shortened or extended by adding or removing resources. A major part of a budget is planned when a tender is created. Many companies intentionally offer lower prices in expectation that a loss introduced during the development phase, will be covered by extra income earned during the project maintenance phase. Budget and time constraints are usually used as a starting point to planning resource allocation.

"Theoretically, we know that we can reduce the overall time schedule by increasing the amount of resources devoted to the work. We also know that each additional resource will make less of a contribution to reducing the schedule, but will still decrease calendar time. This is referred to in

Economics as the Law of Diminishing Returns; each additional increment contributes less to the whole" (Ward 2003).

Adding resources will not reduce overall expenses, but rather will increase expenses, and can quickly empty a planned budget.

"There is also a point when adding resources will not contribute to reducing schedule, but can increase project schedule, and empirical evidence has shown that the optimal number of resources is the square root of man months" (Ward 2003).

Frederick P. Brooks long ago demystified estimations expressed in man-months (Brooks 1995), and besides accusing management of optimism and false assumptions that all will go well, explained that effort and progress are two different categories.

> "The second fallacious thought mode is expressed in the very unit of effort used in estimating and scheduling the man-month. Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and month are interchangeable". (Brooks 1995)

However, using estimations in man-months survived this critic, and it is generally accepted that an estimation failure below 20% of the total estimated time or budget is a good estimation. We wonder how it can be acceptable that a project estimated to cost 1,000,000 USD can be considered a success if it ends up costing less than 1,200,000 USD. Most people complain loudly if an error costs them 10 USD, and for 100 USD many are ready to fight.

For project constraints, estimations use many different models, such as Function Point, Constructive Cost Model (COCOMO), Program Evaluation and Review Technique (PERT), Weighted Micro Function Points (WMFP), Constructive Systems Engineering Cost Model (COSYSMO), and Use Case Points (UCP), among others. Extreme Programming introduced The Planning Game estimation method, and agile development uses Planning Poker- and Story Points-based estimations. (Larman, 2005)

What is interesting for this paper is that in the case of estimation, implementation technology and tools are very important factors, and the choice of technologies and tools can have a significant influence on a project's success. It can be argued that the differences between different technologies, for example Java and ASP.NET, are easy to spot; however, what is not easy to determine in advance is whether a certain feature required by future software application already exists in the libraries or will need to be coded from scratch.

The tools are very important. It is easy to understand that writing code using a simple text editor and command line compiler and linker to build the executable is so far less productive and effective than using an IDE framework such as Microsoft Visual Studio or Eclipse. Proper tools, and combination of compatible tools, can replace a lot of manual and repetitive work, and release the human aspect from error-prone and tiresome tasks that in fact do not add much new value. Tools can do a better job and improve overall quality. Automated tasks release the extra human power for other creative tasks, reduce the number of errors, and can be very important factor for project success.

From a customer's point of view, the project deliverables, in this case application software or system software, must meet functional and non-functional requirements. These requirements can include security, performance, robustness, maintainability, scalability, availability, user friendliness, and generally meeting customer expectation. Meeting customer expectation and application robustness are called quality, and aside from measuring quality by quantitative metrics, such as

number of defects per thousand source code lines, satisfaction can be also quantified and expressed as Excellent, Very Good, Good, Bad, and Very Bad quality.

Implementation Process Overview

Figure 4 illustrates the Implementation processes:

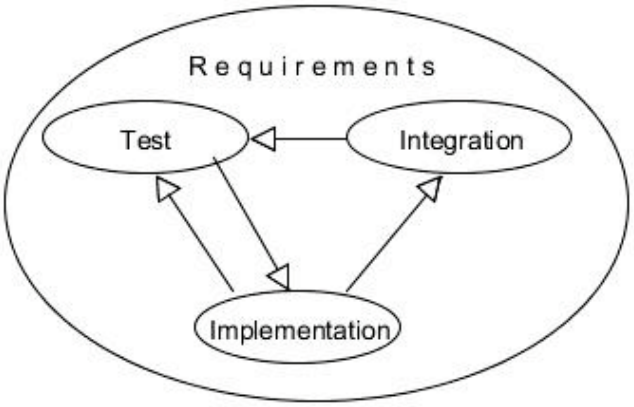

Figure 4 "Implementation Process Description"

Implementation delivers the source code to Integration. Integration generates the executable. Test uses the executable for validation and/or verification. The Test process does not require delivering a new executable. The Test process can use the old executable. When a new version is generated by the Integration process, feedback to the Implementation process is delivered through the Test process. The purpose and scope of all these processes is limited by Requirements.

Implementation Tasks and Workflow

In Figure 5 we have illustrated the Implementation process from Figure 4, and Implementation process tasks and workflow:

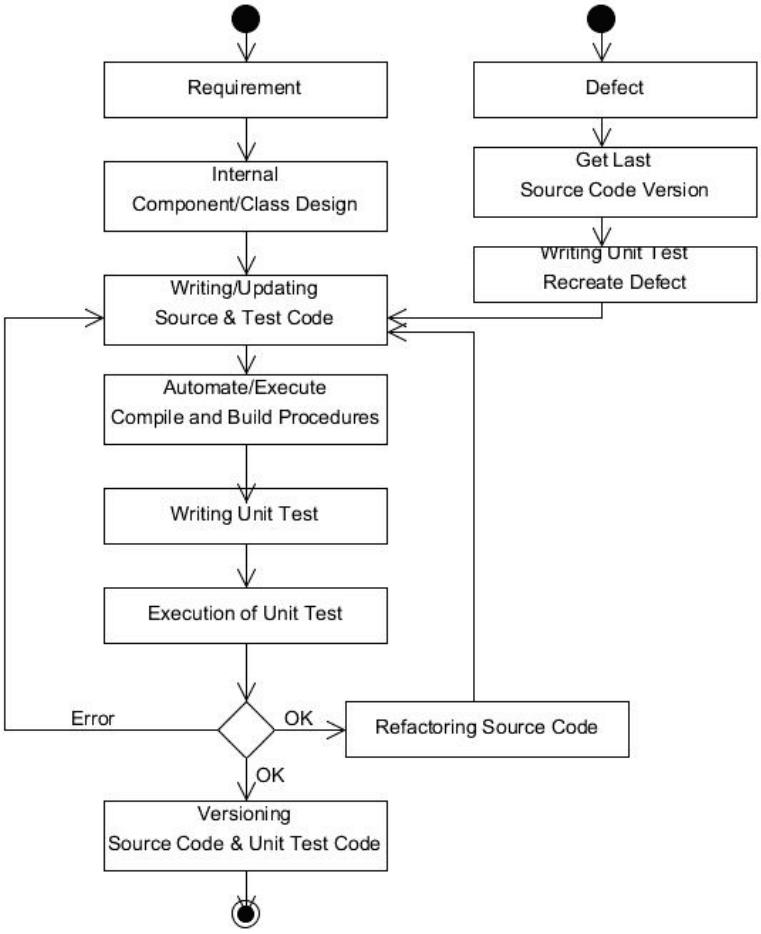

Figure 5 "Implementation Tasks and Workflow"

The Implementation team is primary responsible for the implementation of requirements, as well as for correction of defects discovered during testing. Unit Testing is not able to discover all of these errors, although tools exist that can help test for missing components and interfaces. These tools are generally known as mock objects and several different mocking frameworks are available on the market, such as NMock, EasyMock.NET, MOQ, or Microsoft Fakes.

> "The most trivial bugs are found at the beginning of the testing period, and the most serious bugs are found last. This is almost a tautology: Module testing uncovers relatively simple logic errors inside individual modules; system testing, on the other hand, uncovers major interface errors between subsystems. The point is that major interface errors are *not* what the programmer wants to find at the end of a development project; such bugs can lead to the recoding of large numbers of modules, and can have a devastating impact on the schedule, right at a time when everyone is likely to be somewhat tired and cranky from having worked so hard for so many months." (Yourdan 2006)

Each of the activities in Figure 5, "Implementation Tasks and Workflow", can be supported by tools and utilities.

Figure 6 groups activities from the Figure 5 into the following categories:

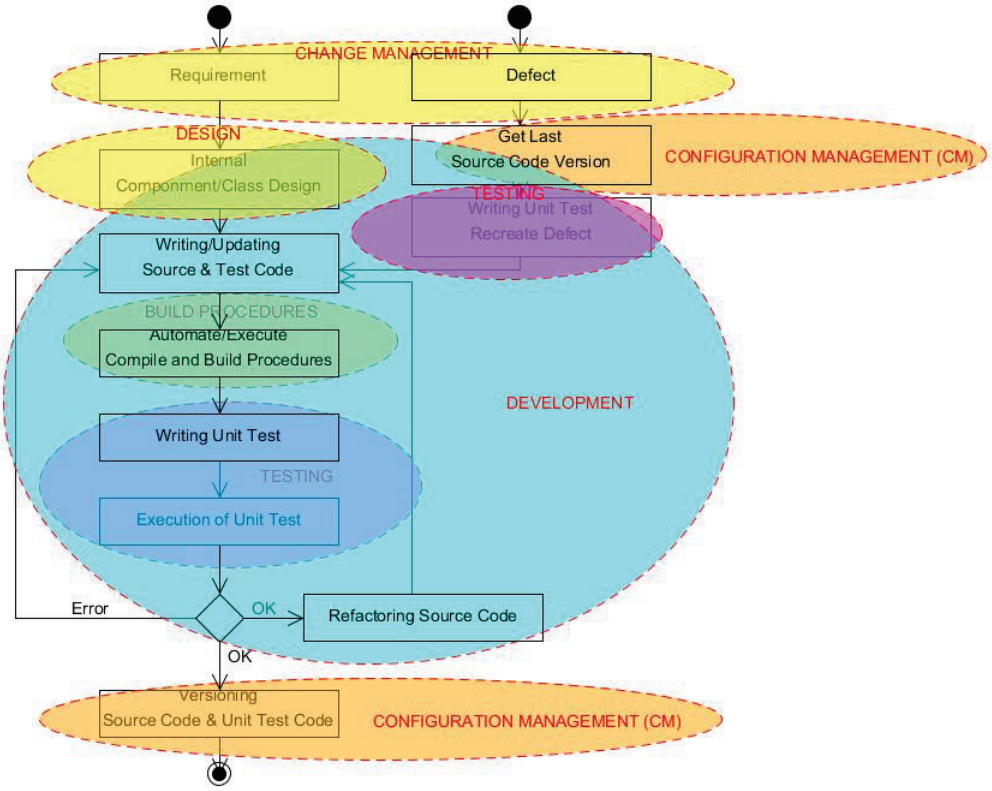

Figure 6 "Implementation Tasks and Workflow Categories"

Categories that are identified in figure 6 are:

1. Change Management (Requirement Management, Defects Management, Work Package, Release Planning, Test Case, Collaboration, Status Tracking and Reporting)
2. Design (Internal Component/Class Design)
3. Development, (Writing/Updating Source Code, Automate/Execute Compile and Build Procedures, Writing Unit Test, Execution of Unit Test)
4. Test (Writing Unit Test Recreate, Execution of Unit Test)
5. Build Procedures (Automate/Execute Build Procedures)
6. Configuration Management (Storing/Retrieving Versioning Source Code, Creating Releases, Retrieving Releases)

Some activities overlap in Development and Test and Configuration Management. This will be explained further and the additional text provides more detailed description of each of these categories ,as well as tools that can be used to improve productivity, effectiveness, and final product quality. These tools are very important in any kind of software development and especially important when using agile and dislocated team development.

Project Management

Managing project schedules and milestones, deadlines, and deliverables, as well as managing project resources, tracking project progress and budget, and communicating and reporting to clients and higher management, and managing quality, are the most important tasks of Project Management.

Project Management tools today implement Web interfaces and this is the preferable solution. It enables all involved parties to exchange messages and read and write notes about particular task.

Microsoft Office Project is software that is particularly dedicated to Project Management planning.

IBM Rational Team Concert (RTC), one of IBM's Rational products based on the Jazz platform, provides breadth, functionality, and covers all needs for agile Project Management. RTC can be used for SCRUM sprint planning, creating backlogs and sprints, as well as for progress tracking.

According to Forester, Rally is the best tool available today for application lifecycle management and the Agile and Lean software development methods. (Grant et al., 2012)

Although vendors claim that their products cover many of the needs for Project Management, it is difficult to find a single product that can satisfy all needs. Project Management needs additional communications tools, such as telephone conferences, video conferences, chats, and file-sharing software, as well as the ability to enable remote access to files stored on the local network, or separate time registration software that is able to track a project's progress and automate invoicing.

E-mail and Voice over Internet Protocol (VoIP) applications, such as Skype, are the most used communication tools currently, because such communications are free and reduce communications expenses radically. IBM Lotus Notes is an integrated suite that contains all communication applications, such as mail, chat, document storage, net-meetings, document sharing, video conference etc. Budget and time tracking, and automatic invoicing require separate application.

Project management without tools is a daunting task and tracking the project's progress is achieved through:

- Status meetings
- Reviews
- Chats and conferences
- Other sessions, for example knowledge transfer, common analyses of requirements or defects, reporting of issues, etc.

Project management and Technical Competence are often twisted together in real life, and very often both aspects demonstrate misunderstandings of the other side's duties and responsibilities. This misunderstanding is also very likely to exist between a developer who is working in the Implementation phase, and those who are responsible for the Build and Deployment phase.

These misunderstanding occur because of differences in the primary focus and task priorities. For example, a developer's primary focus and priorities are project functional requirements, while the technicians who deploy a project are looking at non-functional necessities such as performance, security, or other operational needs like automated deployment/configuration/fallback procedures based on script execution.

In Project Management it is very important to have clear messages about goals, tools, estimations, deliverables, milestones, and deadlines, as well as have pre-conditions ready, such as project support groups or tool support groups, before a new cycle of developer work starts.

Effective Project Management requires a complex and specialized tools such as IBM Rational Team Concert, Rally, or HP Quality Centre.

Change Management

The Change Management's basic role is to register and track all requirements, as well as uncover any defects. For Change Management registration a common mail address box can be used for

communications related to requirements, changes, and defects. The same can be accomplished by using an Excel sheet or a simple Word document, but these are very ineffective and time-consuming modes to track and manage changes. If using a word document, for example, managing a change history of any particular requirement or defect can be a challenge, and reports based on manual inspection can be error-prone. Maintenance of such document/documents can waste a lot of time and resources.

Effective Requirement and Change Management requires complex and specialized tools such as IBM Rational Team Concert, Rally, Borland Caliber, or HP Quality Centre. These are the top tools currently available for Application Lifecycle Management (ALM).

These tools can be used for:

- Requirement registration
- Registration of Change Request and Defects
- Communication to/from clients
- Communication between project team members
- Communication to Project Management
- Analyses and Estimation-related tasks
- Assigning ownership
- Progress tracking
- Release/iteration planning
- Reporting

While maintenance of storing requirements, changing requests, and identifying defects in a word processor document or spreadsheet can be daunting task, storing them in the database can significantly improve this process, especially in team collaboration. Searching, filtering, and reporting in database solutions can be very effective and save a lot of time, as well as significantly improve report quality.

Design

Universal Modeling Language (UML) is still a standard tool for software design. UML can be used for internal component design, as well as for class and methods design. The following is the UML definition available at IBM's Web page ("Unified Modeling Language," 2012):

"***Unified Modeling Language™ (UML®)*** *is a visual language for specifying, constructing, and documenting the artifacts of systems. Complex software designs difficult for you to describe textually can readily be conveyed through diagrams using UML. Modeling provides three key benefits:*

- *Visualization*
- *Complexity management*
- *Clear communication*

You can use UML with all processes throughout the development lifecycle and across different implementation technologies."

Available on the market are open sources or gratis tools such as UMLet, (UMLet 11.5.1 2012). All drawings in this document are made using the UMLet tool.

IBM and Microsoft both offer different tools that support UML. IBM delivers UML tools as part of the Rational Software product family, Rational Software Architect (RAS), Rational Developer,

and Rational Software. Rational Software is expensive and a substantial product that covers a broad spectrum of software development needs and should be tailored to the particular project's needs. Rational UML tools, such as Rational Rhapsody, are able to generate C/C++ and Java source code. ("Generating Code from Rhapsody Model" 2012)

Many of the UML tools are incorporated into Integrated Development Environment frameworks, such as Eclipse and Microsoft Visual Studio .NET.

Microsoft offers Microsoft Office Visio, which can be used to draw UML diagrams. Microsoft Visual Studio .NET publishes an integrated version of the same product that is able to reverse engineer source code to UML diagrams, if the source code was developed in Microsoft languages, C#, Microsoft Visual Basic, or Microsoft Visual C++. ("About Reverse Engineering Code to the UML" 2012)

Development

The definition of Development can be very broad, and in this case we assume that Development is:

1. Writing a source code
2. Compiling and building the executable
3. Executing test
4. Debugging

All of these functions and even more are available today in the many different products collectively known as Integrated Development Environment (IDE) frameworks. Some of these frameworks are more and lees specific for Java language-based development and some are specific for C/C++ and C# and Visual Basic languages.

Many IDE frameworks are currently available and the most popular are:

1. IBM Eclipse
2. Microsoft Visual Studio
3. Oracle NetBeans
4. Oracle JDeveloper

All of the above IDE frameworks except Microsoft Visual Studio are free or open source tools. Microsoft also has a free edition of Microsoft Visual Studio called Microsoft Visual Studio Express Edition. The Express Edition is missing some functions that are available in the full, licensed versions.

IBM Eclipse, Oracle NetBeans and IntelliJ IDEA are the most commonly used IDEs for Java-related development. IBM Eclipse is a highly customizable integrated development environment and has a huge number of plug-ins that extends this tool's functionality, and provides integration to version control systems, unit test frameworks, application servers, profilers, build tools, databases, modeling tools, performance analysis tools, and many other third-party tools.

In cases when Microsoft Windows technologies are used, Microsoft Visual Studio provides tools for .NET family products and IDE frameworks for development based on Microsoft .NET languages.

Build and Deployment

The Build procedure assumes the compiling and building of executables. Deployment assumes the software installation and configuration. Build and Deployment can be specific to a particular environment. If these two processes need to be automated, then these processes need to be exe-

cuted by script. Configuration and changes related to a specific environment need to be configured and updated by a script.

Why it is important to make updates using a script? The most important reason is that a script once tested will execute the same job again and again exactly the same way. Manual changes depending on human beings are error prone. A script, created and tested by experienced professionals, can be successfully executed as many times as necessary, even by those who do not understand what the script is doing.

Another very important reason to use script is that the same script can adapt configuration for different environments and can be used in Development, Test, and Pre-Production and Production environments.

Today the following scripts and scripting languages are most often used:

1. ANT
2. Maven
3. Python
4. UNIX Shell Script
5. Microsoft Power Shell Script
6. Java Script
7. SQL

These script languages are used for software compiling and building executables, software configuration and deployment, configuration of application servers, and database and system software configuration. Scripts are best for replacing manual and error-prone work with an automated procedure. When this procedure is tested and successfully applied in Development or Test environments, the same procedure can be used in other environments, as for example Pre-Production and Production environments.

An important part of build procedures is continuous integration. The following definitions are borrowed from a Martin Fowler article (Fowler 2006):

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily—leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

Martin Fowler (2006) described the practices of continuous integration as:

1. Maintain a single source repository
2. Automate the build
3. Make your build self-testing

Continues Integration can be implemented as a combination of job scheduling and automated build script execution. In this case, build self-testing is implemented by automated test execution. Currently available tools on the market that can do automatic integration and execute automated tests include:

- Hudson
- Cruise Control
- Build Master

Hudson software is free and supports Subversion, SVC, and Clearcase source versioning systems, and is able to execute different scripts, such as ANT, Maven, or Shell Script.

Testing

Test is very important, expensive, and time- and resource-consuming. Test activities directly affect software quality. Test is also an important part of software development process, and any kind of software change requires a test to validate the change, as well as regression testing to ensure that changes did not affect existing functionality. The final software product needs to go through extensive testing and the "V Model", test verification and validation, describes the following test levels (Black 2007):

- Unit Test
- Integration Test
- System Test
- User Acceptance Test (UAT)

A verification is defined as "(1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Contrast with: validation. (2) Formal proof of program correctness." ("IEEE Standard Glossary of Software Engineering Terminology" 1990)

The IEEE Standard 610.12 defines a validation as "The process of evaluating of system component during or at the end of the development process to determine whether it satisfies requirements. Contrast with: verification." ("IEEE Standard Glossary of Software Engineering Terminology" 1990)

Single developers write and execute Unit Tests, and the purpose of this kind of test is to assess a piece of written or changed code to confirm that it works according to specifications. Unit Tests are usually stored in common files and each developer on the team is able to execute Unit Tests written by other developers. Unit Test Suite is a collection of Unit Tests that is used for testing particular methods, classes, or functionalities.

The number of Unit Tests can accumulate very quickly and applications can suddenly have hundreds of test cases that should be executed each time the software application is changed or released. This kind of testing is called Regression Testing and its purpose is the validation of existing functionality and data quality. This can be a daunting task if these test cases need to be executed and verified manually.

Fortunately there are free and open source tools available that automate this very important process of software development. Today mostly code-driven testing frameworks are used, known as the xUnit frameworks collection. These are open source solutions based on the Kent Beck design of SUnit, Smalltalk test tool. Kent Beck and Erich Gamma ported SUnit to Java and called it JUnit (Beck 1989).

Different versions of this automated test framework are available for different programming languages, as for example NUnit for .NET and CppUnit for C++.

The most important advantage of these test frameworks is the opportunity to execute a whole regression test suite at once by executing a class where all these tests are stored as separate methods. Each test can share common preconditions for successful test execution, and in the xUnit jargon this is called a Test Fixture. Tests can also share clean-up procedures that fallback changes, and returns tests to the initial state. Assertions are functions that test a result of each test execution. Different assertion functions exist that can test positive results as well as failures. The expected result is usually sent an input parameter to assertion function, and can be hard coded to compare against the actual result received after test execution. Results of this comparison are depicted as true or false, and also the test result as success or failure. The results are also presented visually, by red and green bars that change color according to the test result.

Today, the most common Unit Test frameworks are:

1. JUnit for Java language
2. NUnit for .NET (C# and Visual Basic .NET, C++) version

Another important testing tool, when a class or interface is not yet available, is a Mock Object. The Mock Object simulates behavior of a missing class, method, or interface and enables the Unit Test case to execute and validate test results. When the missing object is available, test cases will use the correct object version with the same execution and validation procedures. A Mock Object implements the same interfaces as a real object does. The most common reason for using a Mock Object is that a real object is still not available. Some other reasons for using a Mock Object are:

1. Test isolation
2. Testing of interfaces
3. Confirmation that class is working as designed
4. Testing of implementation of error handlings

Mock Object frameworks available on the market include:

1. EasyMock
2. JMockt
3. Moq
4. NMock

Configuration Management

Configuration Management is a very broad and complex subject. This paper will describe only a Source Code Versioning, because it is required by the paper's model presented in the section titled, "An Effective Software Development Environment Model Example".

Source Code Versioning keeps track of any source code changes and each saved change gets a new version number. Source Code Versioning is able to tag releases and branches, and report changes in cases when two different versions or tags or branches are compared, as well as restore any previous versions.

This kind of tool is very important because it enables archiving, tracking, and restoring any changes that are made during the software development process.

Build Procedures and build scripts are usually connected to a Source Code Versioning and automatically check out the latest version, build all necessary application executables, and tag build versions.

Both free Open Source Code Versioning tools as well as proprietary tools are available on the current market. Most popular are:

1. Subversion (SVN), next generation of CVS
2. Concurrent Version System (CVS)
3. IBM Clear Case – proprietary

Social and Cultural Differences

It can be argued that all of the above-mentioned concepts and tools can be applied to any Software Development Project, but there are some fundamental differences that dislocated team development brings up. These dissimilarities are caused by the social and cultural differences in a team that can affect any process and final product quality.

The cultural differences between Western and Asian countries can be huge. For example, women's rights and treatment vary greatly among cultures, and their freedom of movement can be lim-

ited in some countries. The requirements of separating women and men within certain locations can affect team building. A hierarchy within a workplace can create communication issues that may seriously affect development process.

Leadership can meet the challenge in places where it is expected that the leader has the answer to any question. In the Western countries, for example, it is expected that workers take initiative and solve issues, but in other countries the whole process can become held up when a worker waits for his leader to give an order or bring back a solution.

These issues can significantly affect effectiveness and productivity and slow down the whole process.

Issues related to social and cultural differences can be solved best by learning about other people's culture, followed by continuous communication about those differences. Long visits, mutual social arrangements, as well as video conferences, and face-to-face discussions can improve mutual understandings and relationships.

An Effective Software Development Environment Model Example

This model is based on the tools and discussion in previous sections. Figure 8 illustrates an effective Software Development Environment:

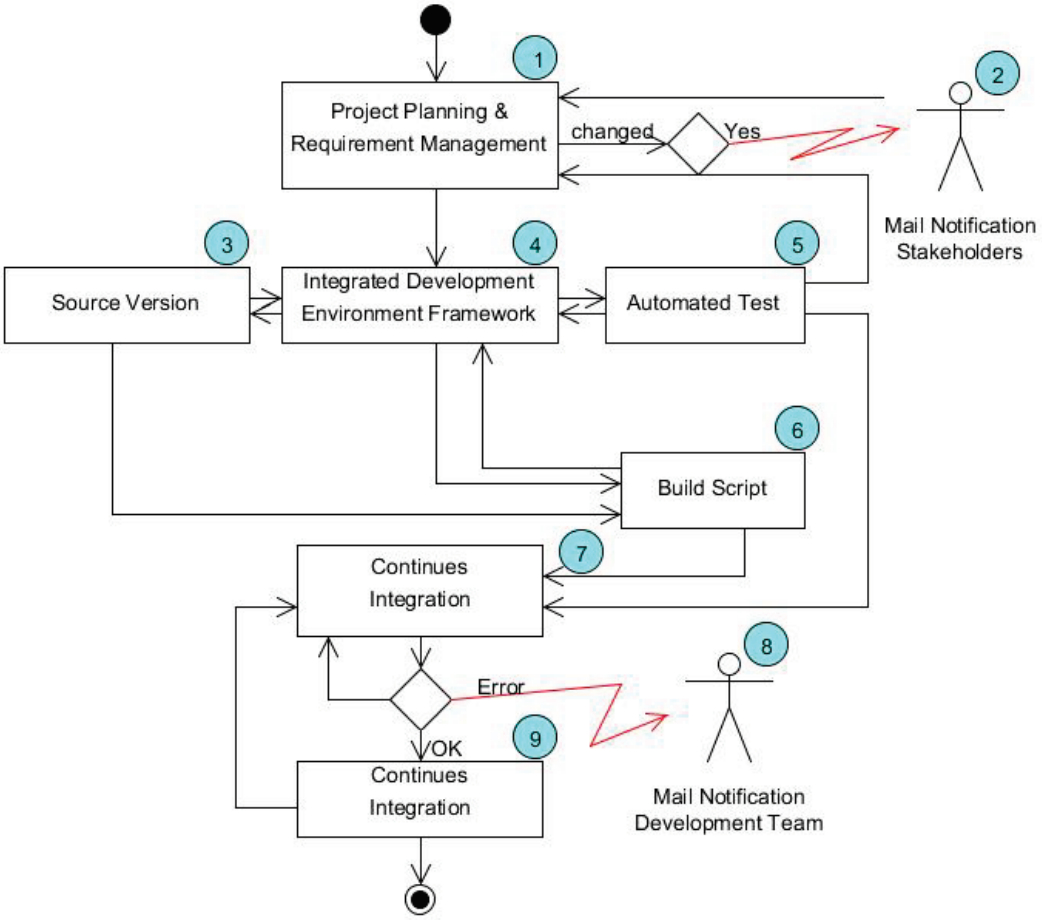

Figure 7 "An Effective Software Development Environment Model"

Incorporated in this model are:

- Project planning and requirement management
- Coding
- Testing
- Build and deployment
- Source versioning
- Team collaboration

Details about each of these processes are described in Figure 8, "An Effective Software Development Environment Model and Tools", and supported by standard functionalities of the proposed tools.

In the above model, the team collaboration is supported by Mail Notification Stakeholders and a Mail Notification Development Team. While members of the developer team are assumed to belong to both of these groups, clients are assumed to belong only to the group of Mail Notification Stakeholders. The simple reason is that it does not make sense to notify clients about build or other errors that occur during development cycles, especially because such notifications usually contain a lot of technical and debug information that can be confusing for non-technical users.

Figure 8 illustrates an implementation of the model as described in Figure 7, and each process is supported by tools that can work together and create an effective and integrated development environment:

This model is created by combining tools discussed in the previous sections.

1 Before development can start, the next iteration should be planned. In the agile terminology that means that first the team needs to decide how long the next iteration will last. In the next step the User Stories from Release Backlog will be selected, which are estimated to implement at the end of the iteration. From selected User Stories the Iteration Backlog is created and each story is assigned to available resources. The User Story probably needs to be broken down into sub-requirements that are described in sufficient detail as required by the development team. Although one can argue that all of this can be described in a word processor, the maintenance and retention of such a document to keep it up-to-date would be a real nightmare and waste a lot of time, Documents change frequently and this same information would be spread over two or more different documents. There should be a record of all communications and defects that have been identified during development, as well as the defects' status, development progress, links to Use Cases and test cases, and design modules.

Another very important step is reporting. An effective report requires a quick overview of the requirement status, task status, defects status, open issues, iteration progress, requirements that are not covered by test case, and many other reports. All these reports need to be created on the fly. An effective iteration planning and development status monitoring requires a proper tool that is able to do all these tasks and even more. Such tools include Rally software, IBM Rational Team Concert, or HP Quality Centre, for example. The IBM Rational Team Concert can be used for projects that utilize different agile methodologies, for example SCRUM development methodology. Hewlett Packard recently acquired the Test Director from Mercury Interactive Corporation and changed its name from Test Director to HP Quality Center (QC).

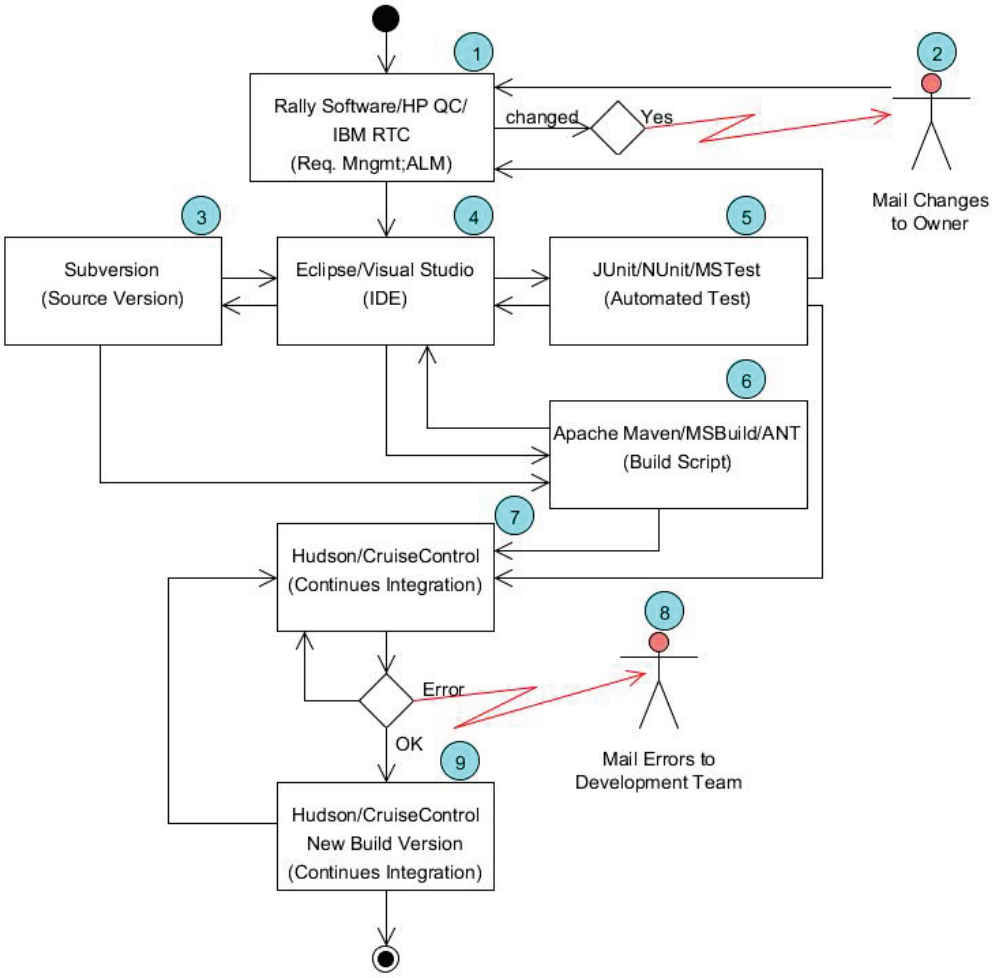

Figure 8 "An Effective Software Development Environment Model and Tools"

"HP Quality Center supports requirements definition and management, release and cycle management, test planning and scheduling, defects management and reporting—all within a single platform with complete traceability driving collaboration between business analysts, QA, and development teams". ("Quality Center Software," 2012)

In HP QC, it is possible to describe a requirement, break down the requirement to more sub-requirements, link the requirement to implementation through work packages definition, define software releases, and create test cases that are linked to one or more requirements. In the HP QC it is also possible to execute test cases in the Test Lab, and create and link eventual Defects. Cross-linked references and reporting supported by dynamically created conditional statements helps the team quickly navigate from requirement to test case and defect. Reports can be textual and graphical. Graphical reports help visualize information about work already done and the remaining work. Reports can quickly discover unforeseen requirements that are uncovered by test cases or not implemented at all. HP QC is a team collaboration platform and Web interface enables the projects team to access and change requirements, add comments, and inspect and execute test cases.

While HP QC offers integrated platform for requirement management, release management, test management, and defect management, IBM offers different tools for each of these requirements.

IBM offers Rational Requisite Pro and Rational DOORS for requirement management, IBM Rational Quality Manager for test and defect management, and IBM Rational Team Concert for application lifecycle management. These tools are integrated and can work together.

“However, IBM provided multiple solutions in some of these areas, leading to confusion about what exactly each of these tools was supporting and how they all worked together. That confusion has largely dissipated with IBM’s current ALM offering. Not only has IBM continued development of its strong suite of products, but it has also stitched them together in a more coherent way”. (Grant et al., 2012)

While HP QC links requirements to test cases and defects, IBM Rational Requisite Pro extracts requirements and use cases from features, and link requirements to use cases and use cases to test case and implementation modules.

IBM Rational tools are integrated into the IBM development tools, for example, Rational Application Developer (RAD), and this development environment enables visual modeling by using UML and accelerates development. IBM RAD generally supports Java technologies, and Web and mobile application development, Java EE, and SOA.

Rally Software, according to Forrester Research, is a leader in Agile/Lean tool providers and Rally Software tools “are optimized for agile planning, project management, status reporting, and other actions that happen within and outside sprints” and can be “less attractive for non-Agile then other general-purpose ALM tools” (Grant et al., 2012).

It is interesting to note that Rally Software currently has only 310 employees and is the Agile/Lean tool leader. The ranking of Rally Software tools by independent researchers, such as Forrester Research Inc. is better than the ranking of tools produced by software companies that have thousands to hundreds of thousands of engineers.

These tools are able to send mail notification to requirement owners and to the other stakeholders when any kinds of changes occur. For example, if it is changed status, description, estimation or priority, added comment, or anything else that is changed, a requirement or defect owner is notified by e-mail. This notification is very importan,t especially in cases of distributed development where sharing information can be an issue. In the above diagram, this mail notification is indicated by a number 2. E-mail notification messages and receivers are a configurable parameter. Automatic mail notifications are a powerful tool for information sharing and distribution, and can save a lot of time spent writing and answering e-mails, which significantly reduce communication overhead. Besides informing stakeholders that a requirement has been changed, a mail message contains all the details about the change. While creating such a message is an easy job for a computer, for a human being this can be a daunting and error-prone task that can take a lot of time, and remove focus and affect creativity, effectiveness, and productivity. Configurable automatic mail notification is a standard functionality in the IBM Rational Team Concert and HP Quality Centre.

When requirements and defects for the next iteration are selected and assigned to developers, a process of writing or changing source code starts. The first step is to check out the last version from Source Control System, in this case from the Subversion. This is indicated by the number 3. If necessary, any previous version can be checked out, or a new version created by combining sources from different versions.

When the desired version is created or checked-in to IDE, particularly in the case of Eclipse IDE or Microsoft Visual Studio, developers can write new source code, and change and refactor existing source code. This step is indicated by the number 4. This process is repeated together with build and test processes indicated by numbers 6 and 5, until the desired functionality or change is implemented.

Writing a source code assumes writing and adding new test cases. The Unit Test framework enables writing of test cases in the same source code as the rest of the software, for example Java or C# code. For Java language the JUnit framework can be used and for C# or VB.NET the developer can use NUnit, or MS Test, Microsoft's newest implementation of automatic Unit Test framework.

Build is a part of the IDE framework and it can be executed as simply as pressing a shortcut key F5 in Microsoft Visual Studio, or even build automatically if this option is selected in the Eclipse. What is the problem here? There is no problem if there is only single developer developing, testing, and checking out/checking in source code in the source versioning control system. This means that build is always using a full version of the source code, because it is assumed that the copy of last version is stored on local drive. This can also be an advantage in cases when there are more developers or a team is involved, because a single developer can test only his or her own piece of code. A problem could occur in cases when multiple developers or developer team/teams are involved during a release build procedure. A release includes all source code developed by all involved parties. A release build can fail or introduce side effect errors because part of the code has never before been tested together with the rest of the source code.

This kind of error can be very unpleasant if discovered late in the process, and can cause delays. If release is delayed, then planned test will be delayed too, as well as all further activities, such as the next iteration planning, start of iteration, and all other activities. When resources are allocated in advance, or customers' resources are involved, this can be extremely unpleasant and expensive. This kind of situation can be avoided if a script is created that can build an integrated version. Such build script shall be able to execute a build externally, outside of the IDE framework, based on the last software version that is checked in the source versioning control system. This script can also be optionally used by developers to test integration build versions. Build script can be developed using Apache Maven, which is an open source project, ANT script, or Microsoft-specific MSBuild script. Apache Maven and Ant scripts are platform-independent and used for Java, while MSBuild is Microsoft-specific and used for build Microsoft Visual Studio projects. Apache Maven uses a common repository to store build dependencies, and enables the sharing of libraries across project boundaries. The Apache Maven repository can store any other kind of object, as for example a file, picture, sound and video, or any other kind of object or library.

Build scripts and source versioning control systems are the most important parts of the Continuous Integration. Continuous Integration uses build script to build executables. Build script is based on the source code that is already stored in the source versioning repository. Build script gets last source code versions from source versioning control system repository, and executes the build procedure. Build script can be made to get any version of source code if necessary to build any previous software version. The execution of build script is a configurable parameter and can be scheduled to be executed every two or four hours, for example, or run on demand. Continuous Integration is not only the build, but rather the execution of the software and the automated test procedures, as well as the test scripts. Build is repeated according to scheduled parameters, and build and test can be executed each hour, twice a day, or as a nightly build outside of the working hours, for example. Jeff Sutherland, SRUM development method founder, described the importance of early test execution in his article, "Ten Year Agile Retrospective: How We Can Improve in the Next Ten Years". (Sutherland 2011)

"At the same time, if software is not done, testing occurs in later sprints. When coaching Palm, Inc. in 2006, we found that one hour of testing at code complete turned into 24 hours of testing effort three weeks later. The impact was that it would take two years to finish features that could be done in a month. Although this might be an extreme case, at Openview Venture Partners, we

use Scrum in the venture teams and in portfolio companies. We have never seen a company that did not take twice as long to deliver software when testing occurred in a sprint later than the sprint in which code was completed." (Sutherland 2011)

Besides executing build script and testing scripts, continuous Integration sofware tools provides the Web interface where it is possible for a team to see all details and reports regarding any portion of the build and test execution. The continuous Integration tool reporting of error can be configured to:

- Report a build error if a build failed
- Report a build error if any of test cases failed

Hudson is a Web-based solution, and is usually run under an Apache Tomcat server. The same server is able to host Java HTML documentation created by the Apache Maven Java-docs plug-in during the build procedure. This means that besides reporting on the build and the success or failure execution of Unit Test, Hudson can be used also as an application documentation server. If a build is successful, Continuous Integration creates an internal release and assigns a release number, as well as storing reports about the build and execution for further use. Standard functionality of these tools is a mail notification in case that build or any of the test cases fail.

If there is any build error, the development team will be notified by e-mail, indicated by the number 8. An e-mail message contains all the details about any failures and other messages, and also all details about the build version where the failure occurred. It is very important to correct any build or test execution issue immediately and not wait. The best practice shows that testing and correcting errors immediately reduces the time necessary for error debugging. This is one of the best approaches when code is developed: write a few lines of code, test, and correct. This is best when complex script is being executed. Early conflict discovery is very important because it is possible to quickly check what has been checked-in last in the source versioning repository and find out what kind of conflict prevented the correct build or test execution.

When a defect or unexpected behavior in an application is discovered, these are recorded in the Requirement Management (Change Management) tools. The solution can be postponed for next iteration or an iteration that is dedicated to defects clean-up. To each defect is assigned an owner, the responsible person, and a status. It can be also assigned a lot of other important information, as for example which build version where the defect was discovered, message details, analysis, expected release planned for correction, comments history, screen shots, links to a module, and many other useful data and information. This can save a lot of future work and avoid confusion, and ensure that it will be taken care and the defect will be followed up on. Search, sort, and reporting tools provide an easy interface to find all kinds of defects. The Defect correction workflow is described in Figure 5, "Implementation Tasks and Workflow", in the "Implementation Tasks and Workflow" section.

Advantages of the Proposed Model

In the proposed model, the only proprietary tools are the Requirement Management tools and Microsoft Visual Studio. All other tools are open source products that can be freely used for development purposes. The IBM Rational Team Concert is free for the first 10 developers, and Microsoft Visual Studio Express edition is also a free product that contains a limited number of functions as compared to the full Visual Studio version.

Besides test and build automation, the above model offers a central repository and highly collaborative environment for storing requirements, defects, test cases, source code, and all other data or information necessary for an effective development team and other stakeholders' communica-

tions. Web interface becomes the standard interface to work with requirement management products.

Ineffective communications, insufficient descriptions, and lost information can create a huge communication overhead and unexpected software product behavior. Sharing and distributing information effectively is a challenge even in small development teams. The Agile development team's magic numbers are between five and eight development team members. This team is large enough to experience many of the issues that ineffective communication can introduce, but is small enough that it cannot afford to use a lot of time updating multiple files. Effective team communication requires that all information be stored in a common, centralized repository, and all updates are done only in one place. Easy access to historical information, previous and current comments and findings, as well as looking at attached screenshots and original messages, can significantly improve understanding and defect tracking, reduce debugging time, and speed up finding proper solutions. Text files and e-mail messages stored in a mail box, although useful, are not sufficient.

The tools can significantly improve the development process. Simple text editors and command line compliers, linkers, and debuggers are replaced by integrated frameworks. Today these frameworks are capable of creating projects, databases, and Web services and descriptors, refactoring and fixing code, and creating missing methods, generating interfaces and HTML and XML files. This is all part of a modern standard development framework's functions.

Development does not consist of only writing and testing one's own code. Industrial projects are often developed by developer teams. To be able to improve the quality of the final product, development requires frequent integration and testing. Frequent integration and testing are time-consuming and error-ridden if executed and verified manually. Both processes can be automated by using scripts and tools, and executed according to predefined scheduling. Another important advantage is that the whole job can be hidden from the team of developers and run individually from the background.

This model also provides a separation of responsibility and each role has a well-defined area of responsibility. For example, the project manager's responsibility is the planning of the releases, monitoring development, and updating progress in RTM. The developers' responsibilities are writing code and test cases and checking out and into Subversion. The system administrator's roles are the configuration and administration of the Subversion servers and repository, and the build manager's responsibilities are the configuration and administration of the Hudson Web server and Apache Maven repository. A system integrator's responsibility is updating Apache Maven build scripts. The current tendency is to create small development teams where the same person executes one or more different roles. This means that one of developers usually overtakes the role and responsibilities of a Build Manager and System Integrator, while other developers can assume a Customer Support role.

Disadvantages of the Proposed Model

Choosing the correct tools for establishing an effective development environment is a challenge. Experimenting with the tools and environment can be a time-consuming process. The development team needs time to become familiar with the tools and development environment. Although most of the development, test, and deployment tools are available for free as open source projects or distributed under GNU Public License, the time taken for learning to use it effectively is not free.

Some required management tool licenses can be expensive. Changing a tool or replacing it with another version can also introduce challenges, such as compatibility issues with other third party tools.

Tools need to be maintained, just like any other application. It means that patches and updates to newer versions are required. If there are multiple tools from different vendors, there is a risk that updating one tool will introduce incompatibility issues to other tools. This kind of risk can be removed by using tools designed to work independent from each other, but then tools integration becomes an issue. However, this is not always possible to achieve.

Manage Quality

If less time is used for repetitive tasks, then more time can be used for code writing and testing. If test execution is automated, then more time can be used for coding. Debugging is a time-consuming process and documenting issues properly can speed up this process significantly, and also improve understanding among team members. Information-sharing and distribution is a very important part of any software development project. If information-sharing and distribution is supported by tools able to send automatic mail notification to involved parties, then it can save a lot of unproductive time wasted by manually writing mail messages, searching for recipients' addresses, analyzing what is different from the previous messages, and creating the message.

Web access to a common repository can significantly reduce communication overhead and keep all relevant information in one place. All stakeholders are directed to the same place where they can see all previous comments and the chronology, check the latest status, and leave comments that are distributed automatically. This kind of functionality is simple to implement yet a very powerful tool for reducing communication overhead, improving communication effectiveness, and reducing misunderstandings.

Improved understanding and wasting less time searching for information releases more time for code writing and testing, and can reduce the number of defects. If less time is spent on defect corrections, then more time is available for development, refactoring, and testing. All of these activities improve the quality of the final product.

Using open source software and tools can save money and the salvaged project budget can be used to assign additional resources to the implementation team, for example, a tester can find time to develop additional test scripts, and report eventual errors earlier. This can significantly improve the quality of the final product, especially if the tester is experienced and dedicated to his job. Another important benefit is the implementation of best practice, when a person other than code writer tests that code.

Automation of repetitive tasks and the promotion of scripting culture are important parts of quality control. When a script fails, it fails always in the same place and it is easy to reconstruct how the failure occurred and what already has been done. If a human being's manual labor fails, then it becomes a major challenge to find out what went wrong, in which order, and what is affected.

The quality of the overall product can also be improved by using another Software Development Methodology, as for example a Test Driven Development methodology where test is developed before the code is written. This approach provides better test coverage, and can deliver less errors and better code quality. This method assumes that all tests are executed by an automated test tool.

Documentation is an integral part of product quality. Good documentation should provide a description of requirements, architecture, and design, as well as describe the technical platform and internal components' design and implementation, and provide end user documentation manuals and guidelines.

Requirements can be extracted from the requirement management and project management tools. Design documentation can be extracted from the source code. Maintenance design documentation in cases of agile development does not make sense. In the event when code design is frequently changed, keeping design documents up-to-date can be an impossible task that can fully occupy

resources and create a huge communication overhead. There are tools that can create UML diagrams, class diagrams, sequence diagrams, dependencies diagrams, and so on from source code. This functionality is built into the Microsoft Visual Studio 2012. UMLet version 12, free version, offers this kind of service too.

Technical documentation can be generated from comments inserted into the source code using tools like the JavaDoc tool for Java code and the NDoc for C# language code. Hudson is a Web-based solution, and usually run under an Apache Tomcat server. The same server is able to host Java HTML documentation created by Apache Maven Javadocs plug-in during the build procedure, and can be used also as an application documentation server.

Although tools can be very important in manage a product's quality, tools are not enough by themselves. Tools need to be supported by methods and processes. In this particular case, if technical documentation quality needs to be improved, then a process must promote the use of time for proper code documenting by writing comments inside the source code, as well as regular monitoring and checking of the generated documentation quality. This can be done by inspecting the Hudson Web site, for example.

Conclusion

The Software Development Environment Model proposed in the paper can significantly improve effectiveness and productivity, and reduce overall costs, as well as improve the quality of the final product.

The section "An Effective Software Development Environment Model Example"-offers a combination of standard tools that do not require much time for setup and administration, and naturally fit to the development tasks. Once installed and configured, these tools can work for a long time without needing to be changed.

The list of advantages of this model can be long, but most important are:

- Team collaboration is centralized
- Automatic build and test procedures
- Early defect warning by continuous build and test execution
- Script-based deployment and configuration
- Mail notification to involved team members in case a requirement or defect changes
- Mail notifications in case a particular build or any of the tests fail
- Automated continuous integration
- Up-to-date and correct documentation, as much as possible

These tools can work fine in the background, and development team members can use them without even knowing any details about installation, configuration, and administration. Developers can be made aware when they get an e-mail message providing all details about error or failure, or when they need extra information; for example, info about the latest builds or when some particular functionality has been executed properly. Developers only need to deliver source code to a Source Versioning system. Other jobs are automatically executed.

All this can be provided by a build script, unit test framework, and continuous integration tools. The continuous integration tools proposed in this model are free, although installation, configuration, and maintenance are not free. All of these procedures require time and resources. The advantages are not small and simple. Early warnings about any kinds of error or issue can be crucial for a project's success, and can significantly affect final product quality. Time used for installa-

tion and configuration can be saved in multiple ways in later project phases. The time used to create a script can reduce the time wasted in manually executing reparative task execution, and reduce the number of errors, overall significantly shortening project time.

The build script, once tested, can save a lot of time for the System Integrator. A build can be created every day and night, or after any kind of code change. The continuous integration tools take over the tester role as well, and this kind of tester is tireless and does exactly that which is described every time, and consistently reports any kind of issue to those who are responsible.

All of this, together with the reduction of development time, can significantly improve the final product's quality. When such a project moves from development to the maintenance phase, it can only benefit from a central repository, where all relevant information is stored in one place. From the scripts it is easy to determine an order of actions and what must be done to execute a job properly. Describing this same information using words is most often a difficult and error-prone job. Understanding and following those words to complete a procedure is even more difficult and error prone.

References

About reverse engineering code to the UML. (2012). Microsoft, available at Internet http://office.microsoft.com/en-us/visio-help/about-reverse-engineering-code-to-the-uml-HP081550745.aspx

Android Developer Tools. (2013). Google Android, available at Internet http://developer.android.com/tools/help/adt.html

Beck, K. (1989). *Simple Smalltalk testing: With patterns*. Available at http://www.xprogramming.com/testfram.htm

Beck K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *Manifesto for agile software development.* Available at http://agilemanifesto.org/

Beck K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas. D. (2001a). *Principles behind the Agile Manifesto.* Available at http://agilemanifesto.org/principles.html

Beck, K. (2002). *Introduction to extreme programming*. IDG, 30-05-2002

Benington, H. D. (1956). Production of Large Computers Programs. *Symposium on Advanced Programming Methods for Digital Computers* sponsored by the Navy Mathematical Computing Advisory Panel and the Office of Naval Research, June 1956. Available at http://csse.usc.edu/csse/TECHRPTS/1983/usccse83-501/usccse83-501.pdf

Black, R. (2007). *Pragmatic software testing: Becoming an effective and efficient test professional.* John Wiley & Sons.

Boggs, R. A. (2004). The SDLC and Six Sigma: An essay on which is which and why. Florida Gulf Coast University. *Issues in Information Systems*, *V*(1).

Brooks, F. P. (1995). *Mythical man month.* Addison-Wesley

Developer resources. (2013). Microsoft. Available at http://www.microsoft.com/en-us/download/developer-tools.aspx?q=developer+tools

Developer tools. (2013). Apple Inc. Available at https://developer.apple.com/technologies/tools/

Eclipse Java development tools (JDT) Overview. (2013). The Eclipse Foundation. Available at http://www.eclipse.org/jdt/overview.php

Fowler, M. (1999). *Refactoring improving the design of existing code.* Addison-Wesley

Fowler, M. (2006). *Continuous integration.* Available at http://martinfowler.com/articles/continuousIntegration.html

Fowler, M. (2005). *The new methodology*. Available at http://martinfowler.com/articles/newMethodology.html

Generating code from Rhapsody model. (2012). IBM. Available at http://publib.boulder.ibm.com/infocenter/rhaphlp/v7r6/index.jsp?topic=/com.ibm.rhp.cg.doc/topics/rhp_c_dm_code_gen_rhp_model.html

Grant, T., McNabb, K., & Anderson, A. (2012). *The Forrester Wave™: Application lifecycle management, Q4 2012*. Forrester Research Inc. Available at http://landing.rallydev.com/forrester2

IEEE standard glossary of software engineering terminology. (1990). IEEE-Std 610.12, IEEE Standard Board, September 28, 1990

Larman, C. (2005). *Applying UML and patterns.* Pearson Education.

Leffingwell, D. & Widrig, D. (2000). *Managing software requirements : A unified approach*. Addison-Wesley.

Oracle Developer Tools. (2013). Oracle. Available at http://www.oracle.com/us/products/tools/overview/index.html

Quality Center Software. (2012). Hewlett-Packard Development Company. Available at http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA1-2115ENW.pdf

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON 26*, August 1970. Available at http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Sommerville, I. (2001). *Software engineering* (6th ed.). Pearson Education Limited

Sutherland, J. (2011). *Ten year agile retrospective: How we can improve in the next ten years*. Microsoft.

Unified Modeling Language. (2012). IBM. Available at http://www-01.ibm.com/software/rational/uml/

Umlet 11.5.1 (2012). UMLet. Available at http://www.umlet.com/

Visual Studio editions. (2013). Microsoft. Available at http://www.microsoft.com/visualstudio/eng/products/visual-studio-overview

Ward, J. A. (2012). The key project constraints. *13 International Conference on Software Quality*, Dallas, Texas, October 2003.

Yourdan, E. (2006). Just enough structured Analysis. Available at http://www.yourdon.com/jesa/pdf/JESA_mgycl.pdf

Biographies

Aleksandar Bulajic is working for IBM Denmark as a consultant and full time employee. He is working more than twenty five years as a computer expert and consultant for some of the top companies in the world.

Aleksandar Bulajic is PhD Candidate at Faculty of Information Technology, Metropolitan University. He graduated from the University of Liverpool, with a Master's in Science degree (Cum Laude) in Information Technology, and from the Economic University with a Bachelor's (BA) degree.

His papers and articles were published in journals and at several international IT conferences in USA, Canada, India, Nigeria, Serbia, Portugal and Australia. He wrote and published several novels and a stage play, and is working on screenplay manuscripts. Besides his professional work and writings, his current interests and writings are mostly related to film and theatre

and interdisciplinary multimedia experiments. He is a member of Liverpool University Alumni Community and Informing Science Institute. E-mail: LANB@45.dk aleksandar.bulajic.1145@fit.edu.rs

Dr. Samuel Sambasivam is Chairman and Professor of the Computer Science Department at Azusa Pacific University. His research interests include optimization methods, expert systems, client/server applications, database systems, and genetic algorithms. He served as a Distinguished Visiting Professor of Computer Science at the United States Air Force Academy in Colorado Springs, Colorado for a year. He has conducted extensive research, written for publications, and delivered presentations in Computer Science, data structures, and Mathematics. He is a voting member of the ACM and is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Radoslav Stojic has a thirty years experience on software development, testing and certification in European aeronautic industry. He has been working on research and development projects ranging from FBW flight control to walking robots and flight simulators. He is currently teaching software quality and testing, artificial intelligence and software for computer games at Faculty of Information Technology at Belgrade.