Software Infrastructure to Reduce the Cost and Time of Building Enterprise Software Applications: Practices and Case Studies

Jalal Kiswani* Muhanna Muhanna** Sergiu M. Dascalu* Frederick C. Harris, Jr.*

*Department of Computer Science and Engineering,
University of Nevada, Reno
Reno, NV, 89503, USA
jalal@nevada.unr.edu dascalus@cse.unr.edu fred.harris@cse.unr.edu

**Department of Computer Graphics, Princess Sumaya University for Technology
Amman, Jordan
m.muhanna@psut.edu.jo

Abstract

In traditional software projects development, there were mandatory activities that should be carried out during the software development life cycle (SDLC). These activities were time-consuming and expensive. They were either performed manually or in basic approaches by different roles. Some of these activities include: version control, project structuring, development environment preparation, software testing, building, packaging, deployment, and management of dependencies including third party libraries and application programming interfaces (APIs). This paper discusses the rationale behind the importance of these activities, and how their automation can reduce the time and cost of software projects, as well as increasing its quality. This paper also consists of some of available open-source resources, and mature free software tools that are already implemented in the software industry of Java technology. Moreover, the paper discusses current limitations of these tools, and includes three case studies of a government and an academic organization, as well as a software development house, at which these recommendations have been implemented.

keywords: enterprise information system applications infrastructure.

1 Introduction

In the software development industry, time is a very important factor for any project's overall success. Reducing the life-cycle time while maintaining or gaining better quality are always significant goals for stakeholders and project's clients[1]. Preparing the proper infrastructure for software projects will support the technical team by avoiding consuming their time on repetitive technical tasks, making them more productive; Such infrastructure can be hardware or software[2].

This paper covers the software infrastructure part, which allows developers to reduce the time spent in technical activities and tasks that are not directly related to the software development, and focus on activities that are more important for the project. This can be achieved by using already developed mature and tested tools, and processes to manage resources and automate many tasks during the SDLC.

Tools and processes discussed in this paper include: (i) version control (ii) artifacts and dependencies management (iii) standard project structure (iv) test automation and (v) continuous delivery and continuous integration, which consists of building, packaging and deployment automation. Although test automation is part of continuous delivery, the authors preferred to discuss it in a separate section.

In any software project, there are assets that should be managed, maintained and tracked [3]; such as documentation, diagrams, and most importantly, the source-code, along with the software packages builds and releases. This process of managing and maintaining these assets is known as Software Version Control (SVC), which is part of the Software Configuration Management (SCM) disciplinary[4].

SVC is not a luxury in the software development industry anymore. The unavailability of SVC may cause a negative impact on the business affecting the software projects overall success [5]. SVC provides

978-1-943436-09-5 / copyright ISCA, SEDE 2017
October 2-4, 2017, San Diego, California, USA

many features, which include the ability to provide an effective way of assets and resources sharing, deletion recovery, conflicts resolving, logical copying (for tagging and branching purposes), among many others.

On the other hand, and based on the "Do not reinvent the wheel" concept, the use of 3rd party libraries and APIs has become an important factor to reduce the cost. However, managing these libraries and their versions had become a serious challenge, which led to introducing the concept of dependency management [6].

In some programming languages and technologies, such as Microsoft .NET platform, creating new projects has never been an issue, since Visual Studio - the official Integrated Development Environment (IDE) of Microsoft - has been the only available option for developing projects in .NET and any other Microsoft technologies. This, however, is not the case in open-source community, with a long list of IDE options for software development, especially for Java technology(e.g. Eclipse, NetBeans, IntelliJ, JDeveloper, etc.). This has become a challenge in terms of projects migration between different IDEs. This challenge can effects either project stability negatively by using different IDEs, or decreases the developer productivity by enforcing unified IDE based on the companys policy.

In traditional software projects and in SDLC, testing is the phase that ensures the quality of software. Testing can be performed manually, by either executing written test-scenarios directly(most likely by the quality assurance or quality control officers), or based on the knowledge of the Business Analyst (B.A.) or domain-expert where they start executing ad-hock test-cases based on their experience and previous knowledge. These manual approaches, however, are not practical in large and long-term projects, because it is time consuming and costly. Test automation is the answer to the above issues, where test cases can be written and implemented as program units using hard-coded values, which validate the functionality against the requirements in a more granular approach. The advantages of test automation over manual testing are that it is only one time investment, and an incremental process. Test automation may run in agile methodology as task testing based, or as a part of a full integrated test for the whole module or system, or both. Although test automation makes the testing phase of software projects more cost effective on the long run, it requires relatively more investment in the short term, since writing and maintaining test cases are most likely performed by test automation developers or technical leaders, which is considered an extra overhead on the project's cost[5].

After finalizing the software development phase, software packages should be prepared, deployed and installed, either for testing or final operation. The packaging, deployment, and installation may be performed in different approaches. It may be done manually - by collecting all the required resources, using script tools (e.g. Apache Ant)- or using proprietary features in IDEs [7]. These methods have the limitations of an expert's availability specific field, along with non-portability between tools, IDEs and environments.

In the software industry and open-source community, tools and frameworks have been developed to automate many activities and tasks required by the development of software's projects, such as: Maven[8], Subversion[9], JUnit[10], and Jenkins[11], discussed in this paper.

This paper is organized into 5 sections. This section covers the introduction, then it is followed by Section 2, which covers the recommended software infrastructure, as well as the background of every activity embedded in the sub-sections; Sections 3 discusses the provided case studies to validate this work; Section 4 discusses the related work of this research; Section 5 concludes the paper and identifies some directions of future work.

2 Recommended software Infrastructure

This section presents the recommended components and tools required as software infrastructure for building high quality software applications, as shown in Figure 1.

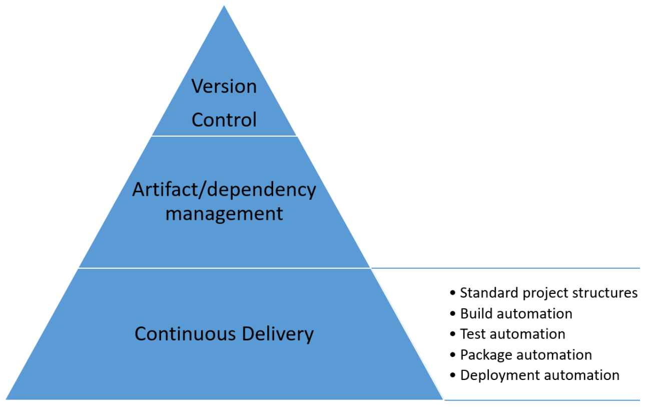

Figure 1: Software infrastructure for software projects

2.1 Version Control

Version control (VC) is the management of software projects assets in general and the source code in particular [4]. It is a combination of tools and techniques that have been used since 1970, to keep track of software assets and its versions, to ensure that it is managed in a reliable and an efficient way.

Most SVC systems consist of multiple commands and functionalities enabling software developers and any other party involved in the managed projects to interact with these assets. Most likely, these assets are stored in a special database called repository. SVC includes features to create, view, update, and delete assets, with ability to compare, retrieve and merge different versions of same asset.

There are two classifications of VC tools and software: Centralized (CVC) and Decentralized (DCVC); currently, two popular open-source tool for CVC is Subversion, and for DCVC is Git.

As mentioned in previous sections, SVC nowadays is important in the software development process. However, deciding between CVC or DCVC in the industry is challenging , because it depends on many factors, include: project scale, team geographical location, team expertise and agility[12].

As shown in Table 1, small-scale projects with relatively close geographical areas and low-level expertise, CVC is a preferred way since it is easier to manage and will reduce the chance of conflicts. However, CVC requires a centralized seniority expert to ensure stability through continuous code review process [12]. In addition, the software development policy in this case should ensure that only low-level and short-term tasks are assigned to developer team by an experienced architect or software designer [13].

Table 1: Factors affecting choosing CVS or DCVC

	Centralized Version Control (CVS)	**Decentralized Version Control(DCVC)**
Team location and distribution	Local	Remote
Project size	Small	large
Team expertise	Low-medium	Medium-high
Task duration	Short	Medium-long
Task nature	Low-medium level	Medium to high level
Architecture and design decisions	Architect /supervisor	developers
Expert supervisor	Highly recommended	Recommended

DCVC, on the other hand, depends on the situation where developers have higher maturity and expertise to handle more high-level and long term tasks [13] and the ability to take architectural and design decisions. Moreover, technical management should ensure the standardization of coding , naming convention, design and architecture decisions, which will result in project's consistency. Failing to do so, in turn, may increase the cost of development and maintenance[14].

2.2 Artifacts and Dependency Management

Another technique and activity that most software developers use is the utilization of dependencies (artifacts) such as third party libraries and APIs to reduce the development time and achieve better quality. Increasing the number of these dependencies may cause a maintainability and stability issue for projects during the SDLC if not managed correctly. Developers and software engineering practitioners have tried to overcome this issue manually using different approaches, such as putting all the artifacts in a separate folder and configuring the integrated development environment (IDE) to use that folder for sharing these libraries with other developers. Sometimes, they upload this folder to a version control system. However, these manual approaches created other issues, such as dependency version consistency, also, it didnt solve the problem of finding these dependencies on the internet for new projects in the long run.

As a solution for the above problems, the concept of artifact repository has become popular, where a special system such as Nexus is used as a repository manager to manage the dependency in an organized and effective way. It is currently the system used for managing Maven central repository, one of the largest software's packages repository on the internet.

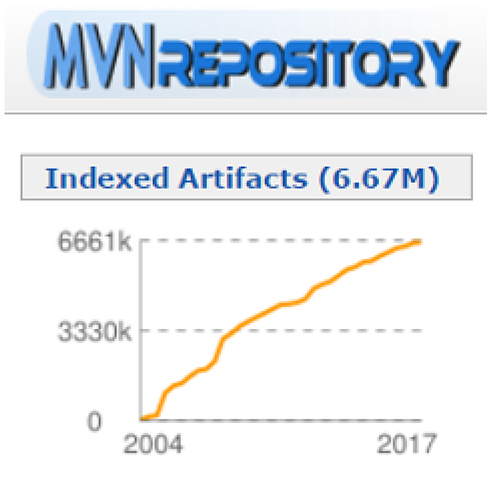

Figure 2: Maven Repository indexed artifacts

The usage of 3rd party libraries, APIs and frameworks in software projects (artifacts) has become a key part of increasing the productivity of developers to deliver software projects successfully and on time [15]. However, manual maintenance of these artifacts is a time consuming and sensitive task, specially when considering version's difference of software projects on development, testing or production environments. Currently, there are many open-source tools that can be used to cover this need such as Nexus. Nexus is used in Apache Maven -the largest open-source central-repository- with more than 6.66M artifacts available as shown in Figure 2 on mvnrepository.com on June 2017. Repository management tools are helpful tools, especially with the availability of public repositories such as Maven central repositories; however, in many cases, there are special needs and requirements that are not available in public repositories, such as: absence of required dependencies, either because of the oldness ,privacy, or licensing issues. Also, sometimes required dependencies may be available in different repositories other than the central one, which causes the project to be manually configured to point to these repositories.

As shown in Figure 3, the proposed recommendation is to implement a software repository manager such as Nexus on the company/organization level; this instance will act as a proxy for Maven central repository and 3rd party repositories, also it can be used to upload 3rd party dependencies, or company internal packages and builds, utilizing the security and privacy features available in these software systems.

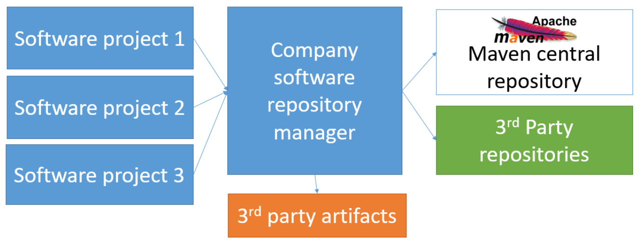

Figure 3: Recommended repository manager configuration

2.3 Software Project Structure

Building software project's structure has become an important task as well. It should be performed carefully to ensure compatibility and migration between different IDEs. One of the mature options for project structure standardization and organization is Apache Maven [16]. Maven has been designed and built based on the concept of project object model, where all the technical aspects of the project are integrated in specific configuration files, enabling portable IDE integration along with built-in packaging and dependency management and lookup [17].

2.4 Test Automation

To increase the quality of software, test driven development approaches have been used for a while, automating unit and integration tests processes. Apache JUnit is open-source tool which is wildly used in different levels of projects. The cost of writing test case, however, is relatively high, especially with mid to large sized projects[18]. Dynamic approach of creating test cases has become mandatory in order to increase productivity and quality, as well as in reducing the cost.

2.5 Continuous Delivery and Integration

The process of building final packages of software applications includes many steps. It starts by collecting the required version of application's source code from the VC, followed by its compiling and packaging , and applying the automated test cases ensuring software's integrity. All these actions and processes form the Continuous Integration (CI) [3]. After package is prepared, it may be published to a dependency repository manager, directly deployed in the designated environment, or both. The full process including the CI is called Continuous Delivery (CD). CD aims to ensure that a high-quality and consistent software package is deployed and implemented. However, it will not be an easy task without a tool that automates all of the above processes [19]. Jenkins is an open-source tool for CI and CD [20]. It is relatively easy to configure and have a big set of plugins required to integrate with many other tools (e.g. SVN, Git, Maven, etc.). Furthermore, It can be deploy applications to production or test environment using different types of plugins [20], such as deployment to JBoss application server using command line interface (CLI) triggered after successfully built process.

3 Case studies

The recommendations and best practices described in this paper have guided the following case studies from different domains with different scalability levels. These case studies include governmental organizations such as Jordan Customs[21], academic such as University of Jordan [22], enterprise software vendors such as International Turnkey Solutions[23]. All of these organizations have implemented most of these recommendations to increase the quality of their systems and the productivity of their developers. In the following subsections, we will briefly describe these case studies.

3.1 Jordan Customs

Jordan Customs (JC) is a governmental organization which is a division of the Ministry of Finance in the Hashemite Kingdom of Jordan (Jordan). JC has a relatively large software development department with more than 30 developers. They have developed, enhanced and supported more than a hundred applications automating their internal and external processes. Part of these applications are a suite that run under the umbrella of the Jordan Customs Financial System (JCFS) which consists of more than 40 modules. JCFS has been developed with a team of 12 software developers from Jordan Customs under the supervision of a software architect and an IT expert, who has been funded by the United States Agency for International Development (USAID)[24]. JC has implemented the recommendations mentioned in this paper including implementing Subversion, Nexus, and Maven. However, due to the lack of time, they were not able to automate the test cases as a part of CI.

3.2 University of Jordan

University of Jordan (JU) is the largest and most popular university in Jordan with more than 40,000 students. As any large academic institute, JU has a large I.T. department with many projects developed in-house. One of these applications is the student self-service mobile applications (JU-App.). JU-App has been developed as part of an agreement and cooperation with a software development vendor specialized in the automation of academic institutes in the Middle-East region called Solid-Soft[25]. JU-app consists of back-end components built on Java and front end-components built on Android and iOS platforms. Same as JC, the development team of the JU-App implemented all the proposed recommendations in this paper except the unit-test automation.

3.3 International Turnkey Solutions

International Turnkey Solutions (ITS) is one of the software vendors for the banking industry with more than 360 banks world-wide on their clients list. One of ITS products is ETHIX-Net, which is a web-based online and mobile banking platforms for both retail and corporate customers. ETHIX-Net previous version has been developed in 2003 with J2EE technology to benefit from the enterprise level features of Java Enterprise Edition, and the portability of Java platform. In 2014, ITS has decided to revamp ETHIX-Net to be based on the latest standards and technologies in the Java ecosystem. For that purpose they out-sourced and hired an expert solution architect to architect, design and lead the development the new platform and system based on the best practices just from the beginning. Based on the long experience of ITS in the software development field and the recommendations of the architect, they implemented the infrastructure preparation list mentioned in this paper, and since they have the availability of a large test automation team, they have been able to implement the test automation mainly for the back-end components.

4 Related Work

To the best of the authors knowledge, no complete reference was found to cover a description or a recommendation for a complete solution to the software infrastructure required for building enterprise software systems. However, there are many publications that cover some individual parts of the proposed and recommended solutions discussed in this paper. These publications were a valuable resources for writing this paper.

For version control, it is relatively not a new topic, as it has been there just from the beginning. In 2009 De Alwis, discussed the main differences between CVS and DVC, he proposed some factors that can be used to decide which to use, including team geographical location and hardware infrastructure such as firewalls[13].

In 2014, Mulu explained the motivations and barriers to move from CVC to DSVC [14]. In 2012, Raemaekers and his colleagues discussed the importance of using 3rd party libraries and APIs, and how it can reduce the development time and cost[26].

The closest research to the work presented in this paper, is the work conducted in the paper "Agile Development and Dependency Management for Industrial Control Systems [15] where authors covered the usage of Apache Maven for build automation, Hudson for continuous integration and Nexus for what they called definite media libraries.

5 Conclusion

This paper presented a recommended software infrastructure for building high-quality enterprise software systems and reducing the lifecycle time and cost. This infrastructure automates many required tasks and technical activities which are not directly related to software project's development. These activities include version control, standard project structure, test automation and continuous delivery. Continuous delivery includes continuous integration and deployment automation, where continuous integration also includes package building and test automation.

Every topic has been presented in details, and the proposed recommendations have been explained with the rational behind them, including a proposed open-source free tools for every activity.

Case studies for many domains including government, education and software development have been presented and discussed as well.

However, during analyzing these case studies, test automation was the missing part, either fully or partially, since technical and development teams thought that it requires a much effort and time, which opens the door for future work and raise the question of: is there an approach of making the development of test automation more efficient and accepted by software developers?

Also, all of the proposed tools and solutions in this paper are Java technology based, so the standardization of the concepts provided by these tools to be utilized with other technologies for different scopes, will benefit both the academia and industry.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant number IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Kurt R Linberg. Software developer perceptions about software project failure: a case study. *Journal of Systems and Software*, 49(2):177–192, 1999.

[2] Paul L Bannerman. Risk and risk management in software projects: A reassessment. *Journal of Systems and Software*, 81(12):2118–2133, 2008.

[3] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk.* Pearson Education, 2007.

[4] Ben Collins-Sussman, Brian Fitzpatrick, and Michael Pilato. *Version control with subversion.* " O'Reilly Media, Inc.", 2004.

[5] Peter Lindsay, Yaowei Liu, and Owen Traynor. A generic model for fine grained configuration management including version control and traceability. In *Software Engineering Conference, 1997. Proceedings., Australian*, pages 27–36. IEEE, 1997.

[6] Yi Cui and Klara Nahrstedt. Qos-aware dependency management for component-based systems. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 127–138. IEEE, 2001.

[7] Alan Dearle. Software deployment, past, present and future. In *2007 Future of Software Engineering*, pages 269–284. IEEE Computer Society, 2007.

[8] Apache Maven. `http://maven.apache.org`.

[9] Apache Subversion. `http://subversion.apache.org`.

[10] Junit. `http://junit.org`.

[11] Jenkins. `http://jenkins.io`.

[12] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering*, pages 36–39. IEEE Computer Society, 2009.

[13] Bryan O'Sullivan. Making sense of revision-control systems. *Communications of the ACM*, 52(9):56–62, 2009.

[14] Kıvanç Muşlu, Christian Bird, Nachiappan Nagappan, and Jacek Czerwonka. Transition from centralized to decentralized version control systems: A case study on reasons, barriers, and outcomes. In *Proceedings of the 36th international conference on software engineering*, pages 334–344. ACM, 2014.

[15] M Mettala. Agile development and dependency management for industrial control systems. In *Conf. Proc.*, volume 111010, page WEPKS001, 2011.

[16] Luisa Hernández and Heitor Costa. Identifying similarity of software in apache ecosystem–an exploratory study. In *Information Technology-New Generations (ITNG), 2015 12th International Conference on*, pages 397–402. IEEE, 2015.

[17] Tim O'Brien and Mountain View Sonatype Inc. *Maven: the definitive guide.* O'Reilly, 2008.

[18] Hyunsook Do, Gregg Rothermel, and Alex Kinneer. Prioritizing junit test cases: An empirical assessment and cost-benefits analysis. *Empirical Software Engineering*, 11(1):33–70, 2006.

[19] Mitesh Soni. End to end automation on cloud with build pipeline: the case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery. In *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on*, pages 85–89. IEEE, 2015.

[20] Valentina Armenise. Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery. In *Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on*, pages 24–27. IEEE, 2015.

[21] Jordan Customs. `http://customs.gov.jo`.

[22] University of Jordan. `http://ju.edu.jo`.

[23] International Turnkey Solutions. `http://www.its.ws`.

[24] Usaid Jordan-Fiscal-Reform project. `http://pdf.usaid.gov/pdf_docs/PA00J9GD.pdf`.

[25] Solid Soft. `http://www.solid-soft.net`.

[26] Steven Raemaekers, Arie van Deursen, and Joost Visser. An analysis of dependence on third-party libraries in open source and proprietary systems. In *Sixth International Workshop on Software Quality and Maintainability, SQM*, volume 12, pages 64–67, 2012.