Evaluation of an Agile Application Development Approach with 4GL Tools in an Offshored IT-Supply Scenario

Dr. Michael Peter Linke
EA Research
Saarbruecken, Germany

The cost pressure for IT-organisations has grown considerably to deliver projects, software, capabilities and features faster and in a more cost-efficient way. Agile software development methods, like Scrum, in cooperation with tools and methods of (partly) automated code generation can be interpreted as an answer to these prevailing challenges. Within this evaluation study in a mixed SAP IT environment a conception for the combined usage of 4GL software tools and agile software development methods together with offshored software developers within different business domains was developed and therefore executed. The results showed that an average cost reduction between 40% and 80% regarding to the overall project setup were within the range of practical realization, if a decent project and communication governance would be in place.

1.1 Motivation and Introduction

The cost pressure for IT-organisations has grown considerably to deliver projects, software, capabilities and features faster and in a more cost-efficient way, and this not only since the financial crisis of the last years. Also the increasing spread of mobile applications for mobiles or smart phones – the number of them actually already exceeds the number of stationary personal computers [Westney1995] – as well as the associated importance of world-wide available “Apps”, contributes to the increased expectations of users on IT organisations. Together with an increasing *consumerization* of IT hardware [Hackenson2008], that is the use of business software on private end devices and vice versa, as well as a likewise increasing mixture of work and life habits, has led to a *21st century customer* [Takeuchi1986] who demands and asks for reactions to inquiries and requests, to some extent also real-time. Agile software development methods, like for instance *Scrum* [Schwaber2002], in cooperation with tools and methods of (partly) automated code generation can be interpreted as an answer to these prevailing challenges; if the work performance is even executed with spatial distance, this complex can additionally be extended by a supplemental, external element, which promises comparative cost methods compared to high income countries on the one hand, and on the other hand requires more communication and standardizing of proceedings, because implicit knowledge [Upadrista2008] cannot be presumed from offshore contractors very often. Here communication governance seems to be a clear advantage, irrespective of the actually established communication tools, and also here there are COTS (Commercial Off-The-Shelf) alternatives of the Open Source Ecosphere available [Bruce2006; Mielnik2003].

1.2 Software Requirements

To generate applications is not for the own sake of IT-organisations, it usually happens on the basis of requests and requirements of the business side, if one disregards the applications of Business Service Management (BSM) [Robertson2006], thus IT-requirements for the IT-organisation itself. How to deal with these requirements qualitatively, from the simple text note up to the use of big modelling environments for effective documentation, is subject of the science of Requirements Engineering [Rup2004], which will only be touched here. Relevant for the single case of application described here are the well known and standardized quality criteria of the IEEE 83, Norm des *Institute of Electric and Electronic Engineers (IEEE)*, published 1998, [IEEE1998] which define basic quality rules for the documentation of requirements in organisations. The criteria formulated within the Standard IEEE 830 claim amongst other things that the requirements are

- correct and valid,
- distinct and free of interpretation,
- •fully documented,
- •consistent and free of conflicts,

- evaluated and prioritised,
- verifiable according to defined criteria,
- granular, as well as
- valid and up-to-date.

Although IEEE standards have high and practical relevance especially in technical environments, the effort for the definition of the above criteria in its entirety for the complete IT-requirements documentation seems to be too high, especially according to many business units who are instructed at many places to create the requirements documentation. Even though IT project practice shows the importance of distinct and clear definitions according to the above mentioned criteria, represented by the still very high number of failed IT-projects where the unclear requirement seems to be the main driver for failure [IEEE1990], in the daily business of organisations it might be an unpopular duty which experiences the more refusal the higher the effort for the target documentation for requirement specifications becomes in fact.

1.3 Agile Software Methods

If one examines the core of many agile software methods, irrespective whether their concrete form is done with a variant of *Extreme Programming (XP)* [Beck1999] or with the *Scrum*-approach [Beck2001], the request of many IT-organisations for faster and more customer-oriented realisation of application development than with the traditional V-models will be understandable. As some examples of agile methods the following ones can be mentioned here amongst others:

- **Pair Programming** (Developer / Tester), the
- **Test-driven development,** or development through
- **Features and /or Story-Cards**

While the requirements definition resp. Requirements Engineering within the V-model - which may be considered as being a classical one - can still be classified as cumulative and ex-ante, then in the agile case this will take place iteratively and ad hoc, in the sense of a step by step Rapid Prototyping. Because of this close time distance between the requirements definition by the specialist division and the visible implementation by an offshore Supply-IT for instance, the observed satisfaction can be increased where appropriate due to the acceleration of the software development, as long as it will be sufficiently integrated into the entire development, governance and review process. Although this seems to be absolutely the case through the use of agile methods [Beck2001] , the implementation of the pure method does not guarantee the success of the project and may also lead to frustrations of the business units because of permanent misunderstanding from the developing supply-IT, for instance if always the same bugs are appearing at reviews or requested features were not implemented permanently.

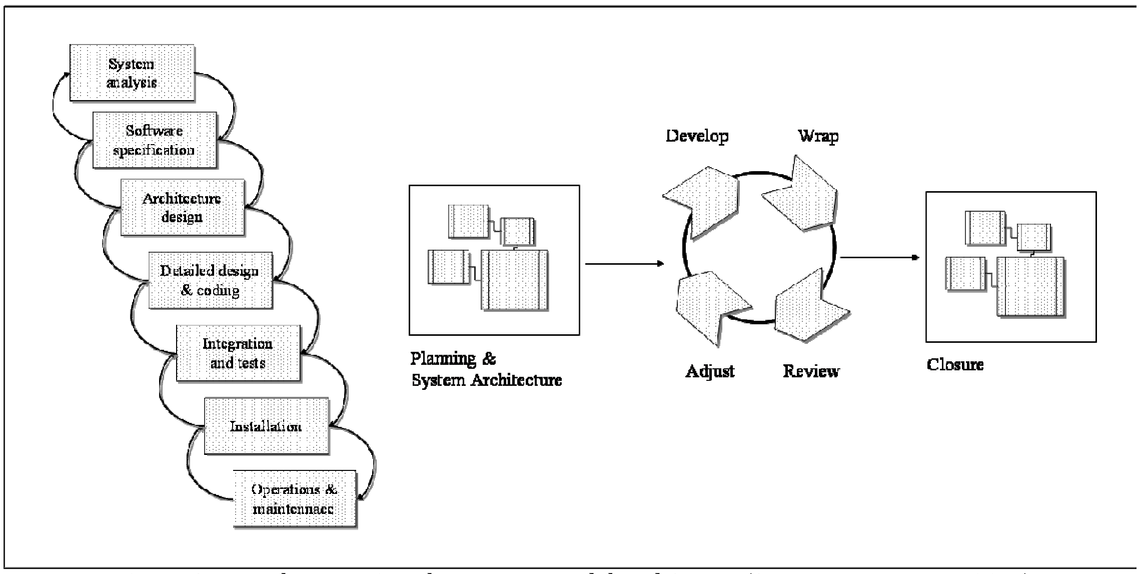


Fig. 1: Strutural comparison between V-Model and Scrum (Source: own composition)

»We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan [...].« [Beck2001]).

Besides, the implementation of these methods seems to be more focused onto the development of new applications, stand-alone where required, and less onto further development and enhancements of monolithic application landscapes as they can be seen very often in corporate groups, like for instance in established Best-of-Breed ERP/CRM/SCM domains [Takeuchi1986]. Especially, if a big amount of interfaces are to be tested or subsystems and other systems are to be integrated, stricter and more process oriented proceedings, which are described among others by the ITIL-process model [DeGrace1990], might be a reasonable alternative to avoid that changes become critical and do damage to the business [DeGrace1990]. Yet, it is to be noted here that especially changes in the sense of agile methods may be considered as characteristic and

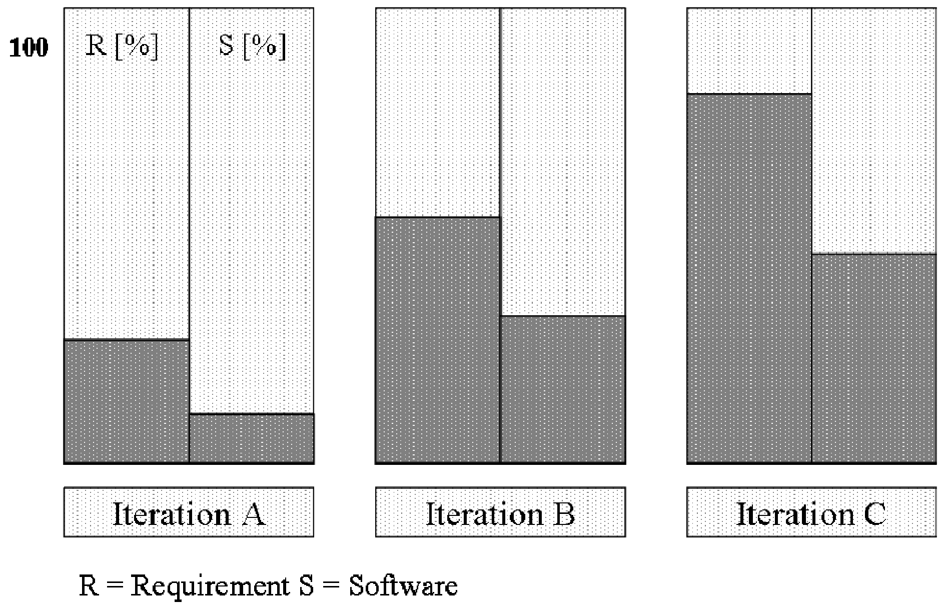


Fig. 2: Distribution of requirements and coded software over several iterations (Source: own composition)

creative elements of this development organisation, so to speak their *Unique Selling Proposition (USP),* in contrast to the change in IT Service Management where it is a minimizing entity to be managed rigidly.

If one accepts these limitations of the agile software development and chooses the scope of IT-development project consequentially, there remains a broad implementation area for this customer-oriented method with the generation of low-interface standalone applications.

1.4 Communication and Collaboration

Thanks to the extensive spread of social media like *Facebook* or *LinkedIn* [Kroll2003; Papacharissi2009] hundred millions of people all over the world are aware of the concrete meaning of social real-time communication. Due to modern and fast web-technologies (like for example an adapted version of PHP for Facebook) [Culnan2010], as well as low entry levels as far as the use of access technologies is concerned (browser) it is possible, to use community and communication features of social media without any problem all over the world. However, the techniques summarized under the keyword "Web 2.0" are not new as far as communication concepts are concerned and may be considered as evolutionary, but unplanned advancements of the *Knowledge Management (KM)*-[Alavi2001] idea of the 1990s, being fully aware of the failure of many KM-projects of the years of 1990 [Levy2009]. Therefore simplicity and ubiquity seem to be of increasing relevance for the omnipresent IT topic mix "Communication & Knowledge".

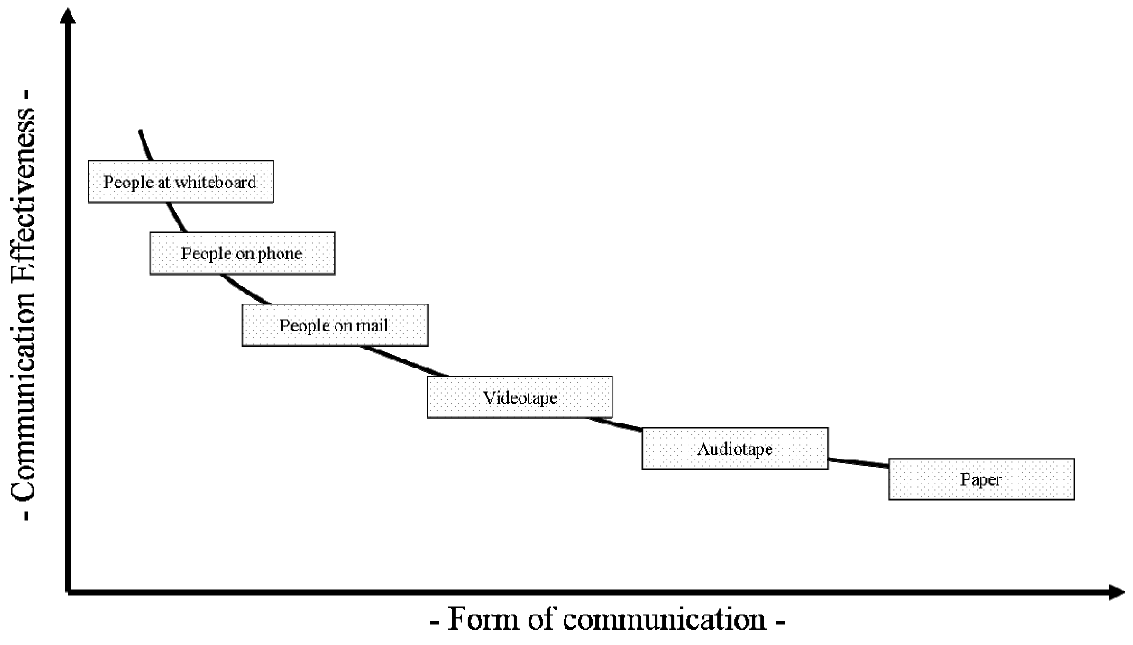


Fig. 3: Effectivness of communication methods

The presence of a communication infrastructure in a software environment could be even more than an enabler in every sense, but more a so-called *Accelerator*, a factor that accelerates the software-oriented total development towards a certain direction. Indications like the rapid *Linux/Open Source* development since the mid of the 1990s [Ljungberg2000], characterized by the participation of millions of software developers at central projects, are a sign that this collaborative and interestingly often beneficial work would not be possible without a single global communication network. To act successfully, especially within world-wide distributed software development projects, tools and techniques are necessary which exceed a normal distributed code (CVS) and version management and which replace onsite meetings between business units and developers so well that friction loss may be minimized.

1.4.1 Forum and Mail-Thread Documentation

In many places an effective email management is not always established to its full extent. Groupware solutions, which in fact offer a broad spectrum of functionality, are used only at a fraction - *Calendar* und *E-Mail* are still the most used modules here. Long-term preservation and especially an easy locating of messages, knowledge artefacts and attachments across years and projects is still a challenge here, which many organisations have to face. Proprietary email storage formats like the PST-format of MS Outlook [Microsoft2007] offer here as local copies only a limited approach to the solution of this problem. Available *Blackboard* and *Forum Applications* [Costello2004; Jackson2007], irrespective of being COTS or Open Source, are offering a solution to this gap with a wide variety of communication, besides of the existing groupware solutions, like for instance *MS Exchange*, *Lotus Notes* or *Novell Groupwise*. Sorted by project, with these tools it is possible to discuss project topics and requirements directly in the set up virtual project rooms and forums, with the possibility of auto-notifiers to all participants of the forum (subscribers). Alternatively, conventional emails of classic email-clients may be copied redundantly to the forum and can there be stored permanently and indexed.

1.4.2 Project and Documentation Repositories

Irrespective of the actually chosen project management procedure model, Prince2 and PMI [Institute2004; Nawrocki2006] may be chosen for example, a central and standardized documentation repository is important and of high relevance for the total project to ensure all project members of the software development environment are talking about the same. CVS Version Management Systems, but also simple applications based on Wiki can be used for a central, web-based storage of project documents.

1.4.3 Instant Messaging and Availability

Technical ad hoc inquiries are of high relevance, especially for distributed teams and agile working groups, quasi to make up for the short personal chat at the office. *Instant Messaging* [Hill2006] and *Presence Aware* [Shaw2007] tools may be implemented to make it possible for distributed teams to discuss and clarify ad hoc topics, irrespective of time and geography.

1.4.4 Bug and Issue Tracking

Especially after going live with a software solution it has proven of value to store all issues, bugs, change requests and incidents in a central database and to make a note about the state of processing. Solutions for bug and issue tracking with different roles, reporting and notifying techniques (email, fax) are the applications of choice to keep records and control even for big projects and to establish any stability within the developed software.

1.4.5 Web Presentation and Audio Conferencing

While with onsite development it is possible to do for instance presentations with Powerpoint slides and live demos of the application in a room with a beamer, it is necessary with a distributed team to switch to other techniques and tools. Web-based *conferencing tools* which could be summarized with the term *Web conferencing* [Angeli2003; Bisdikian1998] offer audio and web presentation techniques for presentations prepared in advance by project stakeholders and project members, but also the possibility for live demos of applications in their particular development phases. An important, additional feature for knowledge documentation in the software development process may be the recording of web- conferences at a defined development phase to comply with possible future legal questions.

1.4.6 Search Engine

If one looks at the different media types that can be used in an IT project (except for the code itself), from a simple office document through email fragments up to Instant Messaging protocols, it will soon become clear, especially for long lasting projects with changing resources, how important it is to find this information during the project quickly and easily. Search Engines, *Full Text Indexer* or *Crawler* [Mertz2001] for the periodical search of different document archives offer the possibility to keep permanent indices of relevant project documents on a sustained basis.

1.5 CASE, 4GL and Code Generators

If one uses the possibilities of agile methods in a software development project and wants to provide an additional plus as far as development speed is concerned, there are methods and tools from the area of *Computer Aided Software Engineering (CASE)* [Gibson1991], partly also transformed to *4th Generation Languages* [Maxwell1990] since the 1990s. These code generators allow the use of integrated work benches, partly automated, simply said: through the massive use of "macros", which the user knows from the office application world, graphical user interfaces, reports, menu structures and database queries and output may be created by drag and drop.

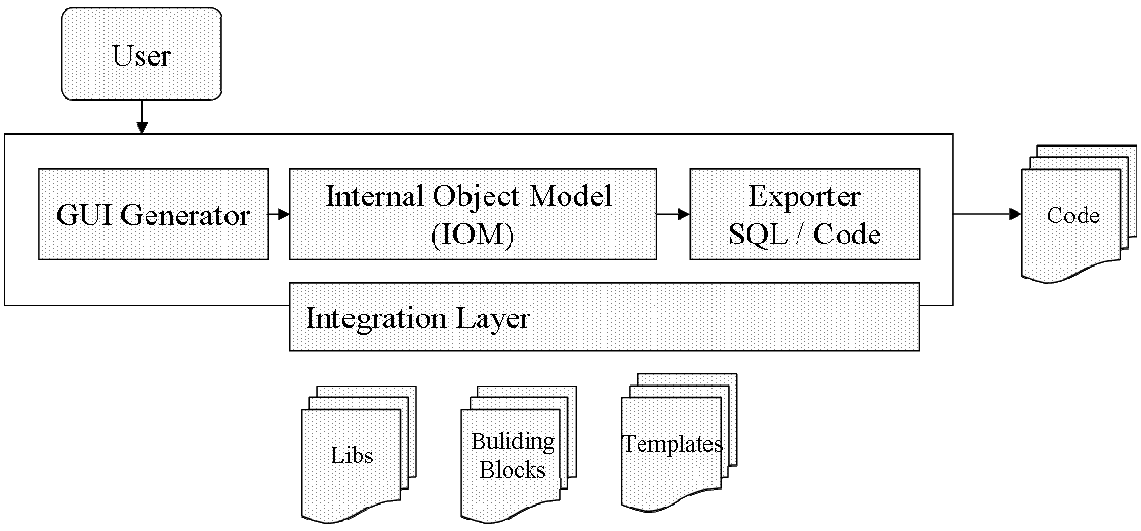


Fig. 4: Structural layout of code generators (Source: own composition)

The code generated by the code generator is usually less maintainable than manually programmed code as the automation mechanism [Royce1987], thus the existing macros and libraries are laid out for universality. Many of the upcoming actions will be managed through parameters in the internal object model.Historically, 4GL languages have seen their widest spread in the 1990s, amongst others in integrated commercial development environments and work benches, where *Gupta* [Vnuk2011], *Informix 4GL* [Maxwell1990], *Clipper* [Hollingsworth1989] or also *TeamDeveloper*

[Corporation2010] have been and still are well known trade names. A low tactical investment protection which was produced by numerous mergers & acquisitions [NetMBA2007] in this segment of the IT industry at the begin of the new millennium have pushed back the spread of this development approach in favour of object-oriented 3GL based languages [Cockburn2006], like Java or .net which allow significantly more flexibility based on individual code, the functional design is left to the human software developer though. However, there are also constructs and libraries for these languages that allow an increasing automation towards 4GL. Whereas with 3GL languages the focus is on the use of standardized control structures. Thus the dividing line often does not exist in the actual syntax of the language, but in the supplied standard libraries and the thereof allowed level of abstraction for programming. Depending on how much use of additional 3rd party libraries is made by a 3GL language with appropriate abstraction and control structures and the less manual coding of the developer is required, the nearer is the approach to the 4GL paradigm, for example in the reporting and with so called *CRUD database applications* (Create, Update, Delete) [Baghdadi2006].

The limits of classifications between the different "generations" are floating for all above mentioned higher programming languages and are not separated neatly and conceptually: while machine code (example: 10110000 01100001) is clearly defined as first generation, the first software code really readable by humans in Assembler (example: mov al, 61h) quasi describes the 2GL which can be found seldom these days, but if found then in device drivers or highly optimised graphical libraries – the high effort for abstractions in programming are usually avoided by developers. Because abstraction requires more CPU performance too, you could definitely try to create a dependency to the increased offer of CPU performance of the last years.

The term 4GL is not defined precisely until today; MARTIN [Martin1982] has tried first to define it in the year 1982 in his book "Application Development Without Programmers", which means the idea of programming without a human programmer. In reality this complete autonomous system couldn´t establish itself until today. Despite code generators and abstraction libraries, the actual program logic will still be created by software developers or at least by technical business analysts. But the 4GL approach has experienced a renaissance with the strengthening of web-based script languages, like Ruby, Python and especially PHP [Sottile2009]. Even IT-laymen can now in the web period design and create web-applications with the support of these technologies.

Besides, many failed 3GL software projects, especially in big organisations, at least have led to take 4GL tools onto the short-list as far as new software developments are concerned. Currently two commercial representatives are the tools “Scriptcase” [ScriptCase2011] based on PHP and “Webdev/Windev” [WinDev2011] based on .net that are propagated also in big IT-organisations. Therefore, there is some hope for one of the two tools to have found an open language standard of 4GL tools for users with the widespread PHP which could reduce the dependency on a single manufacturer, irrespective of the code quality of the generation and this could increase the investment protection in the medium term which in turn could result in an utilization and protection spiral. Code quality and investment protection could therefore be closely connected.

1.6 Evaluation Project 4GL in a Big Organization

On the basis of a project support for a big corporate group the possibilities and prospects for the adoption of 4GL tools in an agile software development project were evaluated to better understand and classify the possibilities and limits of this approach.

1.6.1 Project Setup and Proceeding

The chosen 4GL-tool was implemented at a big corporate group (made anonymous here) within 18 months for seven software projects in the areas Customer Service, Sales and HR. The development control was done by experienced senior project managers onsite, the actual development was done remotely for cost reasons by junior offshore developers who usually worked at times of the day when the onsite project managers were not available. The communication was done in an asynchronous manner through mechanisms described in the appropriate chapter. Radical cost optimising was the aimed business focus; the overall costs were at a very low 5-digit range for seven projects. The servers were hosted externally at a mass host and connected via DSL-lines.

The applications were all new and were based mainly on CRUD-requirements as well as CSV-exports and imports, amongst others to *SAP FI/CO* systems. Six of the projects were completed successfully and within the budget with a maximal delay of 2 months, one project was cancelled by the business unit. The support will now be done by an external supplier, after the displacement of the offshore employees. Compared to the traditional V-model of software development a 7-digit cost saving could be assumed.

1.6.2 Open Source Communication and Collaboration

To allow communication between the offshore team and onsite project managers Open Source Collaboration Tools were implemented to save license costs effectively. A project archive with *Dokuwiki* (about 12 GB project content) [DokuWiki2011] allowed for a central, web-based storage of manually versioned project documents, the infrastructure on application side was built by a forum software with auto email notifier-features based on the PHP-tool *PhpBlackBoard (PhpBB)* [phpBB2011], (about 8900

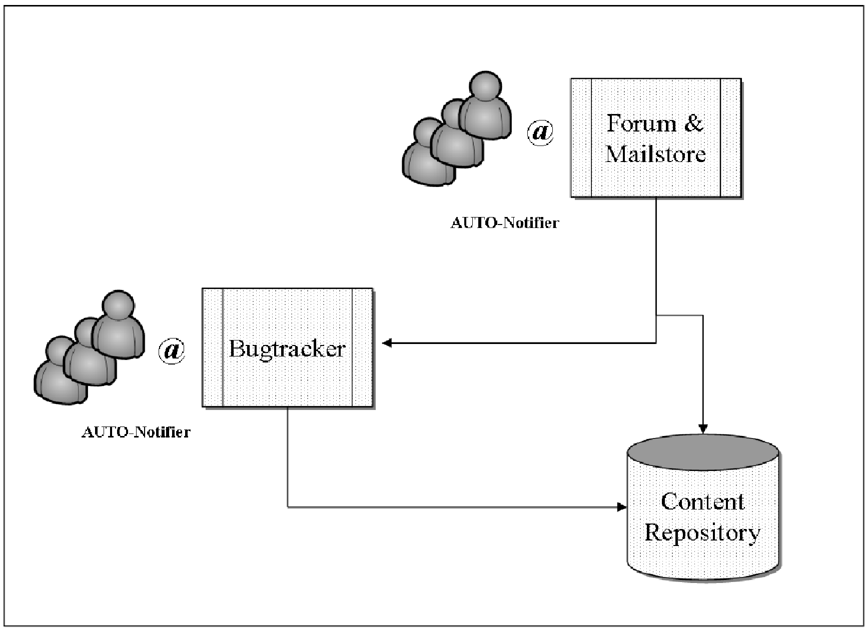


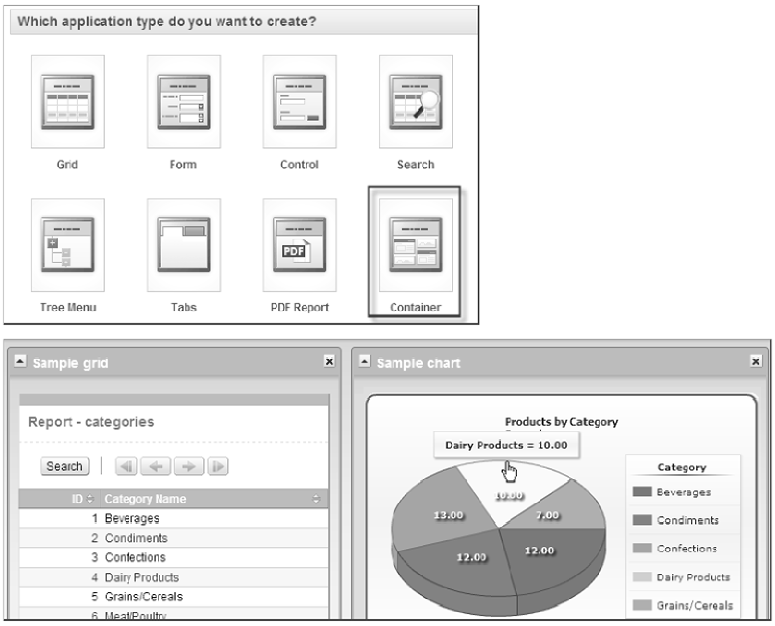
Fig. 5: Collaboration and Communication Infrastructure (Source: own composition)

Messages), as well as a PHP-based bug tracking system *Mantis* [MantisBT2011].

Thereby requirement documents were uploaded in Wiki, the project community was informed via manual news entry in *PhpBB* about the new document. Mail threads from the email client were documented mainly in the forum during the project. For project members who were leaving the access rights were removed promptly, although the entries remained in the forum. Written change requests in the form of paper, email or documents were stored in the project repository, a new bug track entry was created and within this entry there was a reference to the project repository. In this way a consistent proceeding for 80% of the cases could be ensured with the clearly communicated goal: all information at one source.

1.6.3 Implemented 4 GL Tool: Scriptcase

The commercial Open Source product *Scriptcase* of the Brazilian company Netmake [ScriptCase2011] is a web-based representative of a 4GL tool, where it is interesting to point out that also the actual development environment IDE itself is based on the so called LAMP-Stack (Linux, Apache, PHP, MySQL - with ZEND Optimizer).



ID	Category Name
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry

Fig. 6: Screenshots of sample applications within the examined 'Scriptcase' 4GL Tool (Source: [ScriptCase2011])

After the creation of the application via drag and drop, parameterisation, as well as possibly manually added own PHP-code and the creation of a database connection, for example with a MySQL-database, an executable PHP-application will be generated by the code generator. Mainly focused on the simple generation of CRUD-applications, this tool offers numerous modules and components (see table below), as well as a comprehensive reporting framework. The final code generated after completion of the development within the multi-user capable software environment is running natively on LAMP-stacks, without a proprietary *ZEND* component.

Features		
Reports	Forms	Calendar
Editable Grid	Master/Detail	Menus
PDF	Flash Charts	AJAX Support
JQUERY Support	Security Management and Role-based Access Control	Rich Text Editor (WYSIWYG)

Fig. 7: Feature extract from 4GL Tool 'Scriptcase' (Source: [ScriptCase2011])

1.6.4 Usage scenarios and limits

The advised development curve within the agile project procedure could be achieved in its entirety; the selected junior offshore programmers were able to develop CRUD-applications after a couple of hours only. However, in the operating phase it is necessary to do the versioning of the software to be deployed individually with this implemented 4GL-tool. Also database changes, which possibly were to be updated, had to be done manually per release by Delta-SQL scripts in each case. The additional abstraction within the IDE because of the object-oriented design of the tool is costing additional performance; but the generation run on standard hardware with good performance. But for screens and forms with more than 30 GUI-elements performance losses were detectable which could be minimized successfully through sub-screens and tabs. However, the difficulty in maintaining and reading the generated code is still to be rated critically; similar as its predecessors from the 1990s, IDE has to be kept version-controlled and on a sustained basis.

1.7 Summary

With iterative software development methods it seems to be possible to develop software faster than with a development method driven by a traditional V-model. If one joins this general development approach with partly automated code generation, for instance through 4GL-code generators, additional comparative speed and cost advantages can be achieved. The use of offshore developers can be asserted as an additional component for IT-project cost reduction in IT-organizations, because usually they have significantly lower hourly rates compared to onsite developers. This thesis could be proved by an evaluation example with the help of multiple example projects in a big organization insofar as the actual cost reduction can be effectively realized by a factor of 50-80. But at the same time the effort for communication between onsite and offsite resources increased significantly, here it was tried to support this at the best through the use of appropriate online collaboration tools. Despite the offshore components, iterative GUI reviews were possible through telepresence and web-conferencing though, as well as the processing of change requests. However, the historically observed shortcomings of the code quality of the 4GL-tools generation became as significant as potential topics concerning the guaranteed future of the operating company as well as the additional versioning of the development environment itself. In summary the comparative cost effects in this empiric example could be achieved to the extent mentioned above, but with a significant increase of the necessary project communication.

References

Alavi, M., and D. E. Leidner. 2001. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. Mis Quarterly 25(1):107-136.

Angeli, C., N. Valanides, and C. J. Bonk, 2003. Communication in a web-based conferencing system: the quality of computer-mediated interactions. British Journal of Educational Technology 34(1):31-43.

Baghdadi, Y. 2006. Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing. Information Systems Frontiers 8(5):395-410.

Beck, K., M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, and Thomas D. 2001. http://www.agilemanifesto.org /principles.html

Beck, Kent. 1999. Extreme Programming Explained: Embrace Change: Addison-Wesley Professional.

Bisdikian, C., S. Brady, Y. N. Doganata, D. A. Foulger, F. Marconcini, M. Mourad, H. L. Operowsky, G. Pacifici, and A. N. Tantawi. 1998. MultiMedia Digital Conferencing: Web-enabled multimedia teleconferencing system. IBM Journal of Research Development 42(2):281-298.

Bruce, G., P. Robson, and R. Spaven. 2006. OSS opportunities in open source software - CRM and OSS standards. BT Technology Journal 24(1):127-140.

Cockburn, Alistair. 2006. Agile Software Development: The Cooperative Game 2006. Addison-Wesley Professional.

Corporation, Unify. 2010. http://www.unify.com/Products/TeamDeveloper/default.aspx

Costello, B., R. Lenholt, and J. Stryker. 2004. Using blackboard library instruction: Addressing the learning styles of Generations X and Y. Journal of Academic Librarianship 30(6):452-460.

Culnan, M. J., P. J. McHugh, and J. I. Zubillaga. 2010. How large U.S. companies can use Twitter and other social media to gain business value. MIS Quarterly Executive 9(4):243-259.

DeGrace, P., and H. Stahl. 1990. Wicked Problems, Righteous Solutions: A Catalog of Modern Software Engineering Paradigms: PrenticeHall/YOURDON Press.

DokuWiki. 2011. http://www.dokuwiki.org/dokuwiki

Gibson, M. L., and C. A. Snyder. 1991. Computer-aided software engineering facilitating the path for true software and knowledge engineering. International Journal of Software Engineering and Knowledge Engineering 1(1):99-114.

Hackenson, Elizabeth. 2008. CIOs May Learn to Find Value in the Consumerization of Enterprise IT. Enriching Communications 2(2):56-57.

Hill, C., R. Yates, C. Jones, and S. L. Kogan. 2006. Beyond predictable workflows: Enhancing productivity in artful business processes. IBM Systems Journal 45(4):663-682.

Hollingsworth, W., H. Sachs, and A. J. Smith.1989. Clipper Processor - Instruction set architecture and implementation. Communications of the ACM 32(2):200-219.

IEEE. 1990. IEEE Standard Glossary of Software Engineering Terminology.

IEEE. 1998. Recommended Practice for Software Requirement Specifications. In IEEE Std 830-1998. New York: IEEE Computer Society.

Project Management Institute. 2004. A Guide to the Project Management Body of Knowledge: Project Management Institute.

Jackson, P. A. 2007. Integrating information literacy into blackboard: Building campus partnerships for successful student learning. Journal of Academic Librarianship 33(4):454-461.

Kroll, P., and P. Krutchten. 2003. The Rational Unified Process Made Easy: Addison-Wesley, USA.

Levy, M. 2009. WEB 2.0 Implications on knowledge management. Journal of Knowledge Management 13(1):120-134.

Ljungberg, J. 2000. Open source movements as a model for organizing. European Journal of Information Systems 9(4):208-216.

MantisBT. 2011. Accessible from: www.mantisbt.org/

Martin, James. 1982. Application Development Without Programmers Prentice- Hall.

Maxwell, D. 1990. Informix-4GL - The promise and potential of a 4th-generation language. Library Software Review 9(5):297-300.

Mertz, David. 2001. Charming Python: Developing a full-text indexer in Python [Cited: 30.01.2011] http://www.ibm.com/developerworks/xml/library/l-pyind.html

Microsoft 2007. How to manage .PST files in Outlook 2007, in Outlook 2003, and in Outlook 2002 http://support.microsoft.com/kb/287070

Mielnik, J. C., B. Lang, S. Lauriere, J. G. Schlosser, and V. Bouthors. 2003. ECots platform: An inter-industrial initiative for COTS-related information sharing. In Cots-Based Software Systems, Proceedings. H. Erdogmus and T. Weng, eds. Pp. 157- 167. Lecture Notes in Computer Science. Berlin: Springer-Verlag Berlin

Nawrocki, J., L. Olek, M. Jasinski, B. Paliswiat, B. Walter, B. Pietrzak, and P. Godek. 2006. Balancing agility and discipline with Prince. In Rapid Integration of Software Engineering Techniques. N. Guelfi and A. Savidis, eds. Pp. 266-277. Lecture Notes in Computer Science. Berlin: Springer-Verlag Berlin.

NetMBA. 2007. Program Evaluation and Review Technique Cited: 30.01.2011] Accessible from: http://www.netmba.com/operations/project/pert/

Papacharissi, Z. 2009. The virtual geographies of social networks: a comparative analysis of Facebook, LinkedIn and ASmallWorld. New Media & Society 11(1-2):199-220.

PHPBB. 2011. Accessible from: http://www.phpbb.com/

Robertson, S., and J. Robertson. 2006. Mastering the Requirements Process: Addison-Wesley, Upper Saddle River.

Royce, W.W. 1987. Managing the development of large software systems: concepts and techniques. Proceedings of the 9th international conference on Software Engineering, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

Rup, C., and Sophist Group. 2004. Requirement Engineering and Management. Munich: Hanser Publishing House.

Schwaber, Ken and Mike Beedle. 2002. Agile Software Development with Scrum: Prentice Hall.

ScriptCase. 2011. http:// www.scriptcase.net/phpgenerator/home/home.php

Shaw, B., D. A. Scheufele, and S. Catalano 2007 The role of presence awareness in organizational communication: An exploratory field experiment. Behavior &

Information Technology 26(5):377-384.

Sottile, Matthew, Timothy G. Mattson, and Craig E. Rasmussen. 2009. Introduction to Concurrency in Programming Languages Chapman and Hall/CRC.

Takeuchi, H., and I. Nonaka. 1986. The new new product development game. Harvard Business Review 64(1):137-146.

Upadrista, Venkatesh. 2008. Managing Offshore Development Projects: An Agile Approach: Multi-Media Publications Inc. .

Verner, June, and Graham, Tate. 1988. Estimating Size and Effort in Fourth-Generation Development. In IEEE Transactions on Software Engineering. IEEE.

Vnuk, Lubos. 2011. Gupta 4GL http://www.sqlweb.vnuk.org/index.htm
##gup/index.htm##articles/oracnt.html

Westney, D. E. 1995. The knowledge-creating company - How Japanese companies create the dynamics of innovation - Nonaka,I., Takeuchi,H., Sloan Management Review 36(4):100-101.

WinDev. 2011. http://www.windev.com/index.html