The Open University of Israel

Department of Mathematics and Computer Science

Application Lifecycle Management Environments: Past, Present and Future

Final paper submitted as partial fulfillment of the requirements

Towards an M.Sc. degree in Computer Science

The Open University of Israel

Computer Science Division

Submitted by: Moshe Kravchik, 314109555

Nachal Lachish 16/10, Beit Shemesh

052-4286046

mkravchik@hotmail.com

Prepared under the supervision of: Dr. Shai Koenig

Submission date: August, 19 2009

Acknowledgements

I'd like to thank my supervisor, Dr. Shai Koenig for sparking in me the interest in the field of ALM Environments. Without his intelligent guidance and tremendous help in finding useful materials this work could not happen. I would also like to thank Dr. Shmuel Tyszberowicz for thoughtful ideas and constructive comments that guided me during this work.

I would like to thank Dr. Reuven Rosenberg for his help in proofreading and correcting parts of my work. I'm very thankful to Evgeny Kurtser who provided me great help in editing this paper and improving its understandability and conciseness.

I'm grateful to Mr. Avi Yaeli from IBM Haifa Research Labs and Mr. Anthony Wasserman for sharing with me their insights on different aspects of the ALM space.

I also want to thank Railways of Israel where I wrote most of this work.

Last but not least, I'd like to thank my family that supported me throughout the 5 years of my studies, gave me a lot of encouragement and inspiration, and eventually made this work possible.

Contents

Abstract 8
1 Introduction 10
2 Background 12
2.1 Basic concepts 12
2.1.1 Application 12
2.1.2 Lifecycle 13
2.1.2.1 Lifecycle definition 13
2.1.2.2 Processes and activities 14
2.1.3 Management 18
2.1.4 Environments 22
2.2 History 23
2.2.1 Evolution of CASE 23
2.2.2 IPSE 26
2.2.3 PCTE 31
2.2.4 ALM 1.0 34
3 ALME Classification Framework 36
3.1 Classification Aims and Principles 36
3.2 Related work 37
3.3 Classification Framework 38
3.3.1 Technical aspects 40
3.3.2 Organizational and Business Aspects 42
4 Case Studies of Current ALME Projects 44
4.1 IBM Rational Jazz 44
4.1.1 Design goals and architecture 45
4.1.2 Jazz Platform Evaluation 48
4.1.2.1 Breadth of lifecycle support 48
4.1.2.2 Integration 48
4.1.2.3 Role-based views 55

4.1.2.4 Traceability and reporting 55
4.1.2.5 Process definition and automation 55
4.1.2.6 Platforms support 55
4.1.2.7 Distributed teams support 56
4.1.2.8 Scalability 56
4.1.2.9 Security 57
4.1.2.10 Incremental implementation 57
4.1.2.11 Integration and interoperability with existing solutions 57
4.1.2.12 Breadth of vendor support 58
4.1.2.13 Success stories 58
4.1.3 Rational Team Concert Evaluation 58
4.1.3.1 Breadth of lifecycle support 58
4.1.3.2 Integration 61
4.1.3.3 Role-based views 63
4.1.3.4 Other technical aspects 64
4.1.3.5 Incremental implementation 65
4.1.3.6 Integration and interoperability with existing solutions 65
4.1.3.7 Breadth of vendor support 65
4.2 Microsoft's VSTS 65
4.2.1 Architecture 66
4.2.2 VSTS Evaluation 68
4.2.2.1 Breadth of lifecycle support 68
4.2.2.2 Integration 70
4.2.2.3 Role-based views 74
4.2.2.4 Traceability and reporting 75

4.2.2.5 Process definition and automation 76
4.2.2.6 Platforms support 76
4.2.2.7 Extensibility and openness 76
4.2.2.8 Distributed teams support 78
4.2.2.9 Scalability 78
4.2.2.10 Security 79
4.2.2.11 Incremental implementation 80
4.2.2.12 Integration and interoperability with existing solutions 80
4.2.2.13 Breadth of vendor support 81
4.2.2.14 Success stories 81
4.3 Comverse's DiME 82
4.3.1 Design Goals and Architecture 83
4.3.2 DiME Evaluation 84
4.3.2.1 Breadth of lifecycle support 84
4.3.2.2 Integration 88
4.3.2.3 Role-based views 90
4.3.2.4 Traceability and reporting 90
4.3.2.5 Process definition and automation 91
4.3.2.6 Platforms support 91
4.3.2.7 Extensibility and openness 91
4.3.2.8 Distributed teams support 91
4.3.2.9 Scalability 92
4.3.2.10 Security 92
4.3.2.11 Incremental implementation 92
4.3.2.12 Integration and interoperability with existing solutions 93

4.3.2.13 Breadth of vendor support 93
4.3.2.14 Success stories 94
4.4 Summary 94
5 Trends and tendencies of ALME`s 96
5.1 Large ALM software vendors 96
5.2 Medium and small vendors 98
5.3 New ALM solution providers 99
5.4 Open source and ALM 100
5.5 Proprietary ALM solutions 101
5.6 Technology trends of ALME`s 102
6 Conclusions 105
Appendix 1: List of Abbreviations 108
Bibliography 110

Table of Figures

Figure 1. The Lifecycle Processes 15
Figure 2. The scope of ALM (adopted from (IBM, 2008)) 21
Figure 3. ALM 1.0 suites (adopted from (Schwaber, 2006)) 35
Figure 4. The developed ALME Classification Framework 39
Figure 5. Jazz server and clients (from (IBM, 2008)) 46
Figure 6.Jazz components (from (IBM, 2008)) 47
Figure 7. An EMF model definition in UML, XMI and Java (adopted from (Eclipse, 2005)) 50
Figure 8. Team Process Customization Dialog 54
Figure 9. Rational Team Concert Components (adopted from (Rational Team Concert Capabilities)) 59
Figure 10. Open Services Resources and Relationships (from (Typical Lifecycle Resources and Relationships)) 63
Figure 11. VSTS architecture (from (Microsoft, 2009)) 66
Figure 12. VSTS Architecture with an Integrated Extension (adopted from (Minium, 2006)) 67
Figure 13. VSTS URI Format (from (Microsoft, 2009)) 71
Figure 14. Visual Studio Extensibility (from (Microsoft, 2009)) 72
Figure 15. Permissions settings of the Agile process template 73
Figure 16. Sample tool registration data 77
Figure 17. Accessing TFS through a reverse proxy (from (Meier, Taylor, Bansode, Mackman, & Jones, 2007)) 78
Figure 18. DiME Architecture (from (Koenig, 2008)) 84
Figure 19. DiME Events 88
Figure 20. DiME Information Model (from (Koenig, 2008)) 89
Figure 21. The DiME Deployment Process (from (Koenig, 2003)) 93

Abstract

Industrial software development is a complex process involving many participants, often geographically distributed, spanning long periods of time and involving the creation and management of large quantities of inter-related information. Due to its complexity, this process must be managed - planned, executed and controlled - in an efficient and predictable way. Over the last thirty years many tools have been developed to support software development planning, execution and control. Currently these tools provide point solutions - each tool specializing in one or two specific areas of the software development lifecycle (e.g. requirements management, test management, defect management). However, such "point" tools are not aware of the other tools that are being used as part of the development effort. This lack of connection between these tools has become one of the major problems of modern industrial software development.

Application Lifecycle Management (ALM) deals with approaches, methodologies and tools for integrated management of all aspects of software development. Its goal is to making software development and delivery more efficient and predictable by providing a highly integrated platform for managing the various activities of the development lifecycle from inception through deployment. Attempts to create such an integrated platform for software development are not new. A number of systems and solutions were developed in the past to address this problem. However, all of them have failed to provide an industrial-strength solution adequately addressing the needs of ALM.

This work includes a systematic study of ALM tools and environments. It provides, as a basis, a definition of ALM based on a number of existing definitions. Two major past initiatives were studied to find the reasons for their failure and the main factors influencing the quality and success of an ALM solution. It appears that the main reasons for the failure of these early integrated lifecycle management systems were process and technology immaturity combined with an inadequate understanding of the integration needs of industrial software development.

The central part of the work proposes a classification framework for ALM environments. This framework is based on the previously defined ALM goals and the lessons learned from the

unsuccessful early initiatives. This framework can be used for systematic assessment of such environments for both practical and theoretical purposes. The framework consists of a set of criteria divided into technical, organizational and business aspects. The proposed classification framework was applied then to the analysis of three contemporary ALM systems belonging to the new generation of ALM environments. These are Microsoft's Team System, IBM Rational's Jazz, and DiME, a proprietary system wholly developed in Comverse.

In conclusion, the work contains a review of the trends and future directions of this new ALM domain.

1 Introduction

Today's software development process goes through multiple stages, spanning a number of months and sometimes years, involving large numbers of people, and managing a very large number of entities. It is comprised of multiple activities which process and control the development artifacts. Major savings both in time to market and cost can be achieved by improving this process.

In recent years more and more organizations understand the need for coordinating the different activities and a number of commercial products address this need. However, the current state of the practice is that tools used in a real life software process are usually very task-specific, come from different vendors, have their own data repository, use different user interface and internal representation idioms, are not aware of one another and are somewhat difficult to integrate (Schwaber, 2006), (Borland, 2007). Quite often general purpose tools, like Microsoft Excel and Word, are used for many of these activities. While integrated environments for programming related tasks are very common, their integration with other parts of the software process is quite weak in most organizations. According to a survey conducted by Borland in 2007, "the majority of organizations are delivering software in extremely heterogeneous environments, with nearly 90 percent relying on multiple tools from several different vendors to get the job done" (Borland, 2007). This survey also found that the interoperability problems between these tools are critical. "More than half of respondents identified one of several `connection-related` issues – disconnected processes, lack of visibility and traceability across the lifecycle, lack of metrics, poor interoperability between tools, functional silos – as the biggest software delivery challenge or deficiency their IT organization needs to overcome" (Borland, 2007).

As a consequence, a lot of efforts, mostly manual, are required to coordinate between different activities and tools during the application development process (Schwaber, 2006). Application Lifecycle Management (ALM) addresses the problem of coordination of these activities, integration between tools and process optimization and automation. The need for such

coordination has been widely recognized by software development enterprises. More than 75% of them are aware of or are already implementing some ALM solutions (Schwaber, 2006). The first (and most currently available) ALM solutions have not been built from the ground up, but have rather emerged from existing tools and provide abilities of tool-to-tool integration between them.

Right now, a new, more fundamental approach is being taken by a number of the largest software vendors, such as Microsoft, IBM, Borland, as well as the open source community and smaller vendors. They are developing a more comprehensive, more open, more powerful solution to the problem. These solutions follow a number of strategies: some (IBM (Jazz Overview), Borland (Borland)) are building platforms to allow interfacing a number of existing tools together, some (Microsoft (Visual Studio Team System 2008), MKS (MKS)) are building a whole new system around a unified repository, some (Serena (Serena Sofware), Compuware (Compuware Corporation)) are building an open ALM platform for multiple vendors to connect to. This new generation of Application Lifecycle Management Environments (ALME) is commonly known as ALM 2.0.

In this work the existing architectural and strategic approaches of ALM 2.0 were studied, identifying their strengths, weaknesses and target audience. Past ALM initiatives that have failed (sometimes now called ALM 1.0) were examined to see what caused their failure, trying to use this knowledge in analysis of today's solutions. In addition, the similar evolution of lifecycle support products that emerged in two other areas: ERP (Enterprise Resource Planning) and PLM (Product Lifecycle Management) (Abramovici & Sieg, 2002) was considered. The purpose was to identify what strategies and approaches are going to be most successful in the ALM domain. The central part of the work was proposing an ALME classification framework and applying it to a number of leading ALM systems. These systems (except for DiME) were installed and used by the author. The conclusions presented in this work are based on technical documentation, available academic and technical reviews of the systems, independent analysts' reviews and personal communication and experience. They do not rely on marketing materials describing those systems.

2 Background

This chapter presents an overview of the concepts central to our topic, application lifecycle management environments (ALME). It covers the application lifecycle itself and the processes and activities that comprise it. A number of definitions of ALM and the differences between them will be presented. Finally this chapter gives a historical overview of the tools and environments providing support for the application lifecycle activities.

2.1 Basic concepts

In order to understand what application lifecycle management environment is we have to define what we mean by each one of its four components: application, lifecycle, management and environment. Each one of these concepts is widely used in different contexts and the sections below will clarify what they mean in the context of ALM.

2.1.1 Application

In the context of ALM "application" usually means a software system or a system where software plays a central role. Other kinds of systems use different terminology for lifecycle management. For example, the term Product Lifecycle Management is used in manufacturing industries such as automotive, aerospace and machinery (Abramovici & Sieg, 2002). While software applications share a lot in common with other types of products, software production has its specificity which is central to ALM. This paper will refer to "software application" from this point forward when the term "application" is used.

Historically, software development was thought of as very different from traditional product manufacturing (Boehm, 2006). Software development was considered to be driven to a large extent by "creativity and inspiration" and thus hardly predictable. This perception led to a major difficulty in delivering software projects on time. In order to cope with this problem as well as the increasing complexity of software products researchers and practitioners started to study the process of software development. In the 80's the software process area began to develop and a

lot of progress has been achieved since. A number of institutions have been established to study the software development process with the most important of them being the Software Engineering Institute (SEI) in the United States and the European Software Institute. The International Standards Organization (ISO) has created an important ISO 12207 (software lifecycle activities) (International Standard Organization (ISO/IEC), 1995), (Singh, 1998).

As Fuggetta points out, the initial understanding that software processes are somewhat unique has caused a major problem (Fuggetta, 2000). The software process community has not taken advantage of the existing research and experiences of other communities. As a result, it had to redo a lot of the research work and to repeat the others' mistakes just to find that a lot of commonality exists. While focusing on the technology, the organizational and social aspects of the software process were often forgotten. Nowadays this mistake is being corrected. Experience shows that in order to be successful, the software process needs to consider the people involved in it and the organization that supports it. ALM is a good example of how the lessons learned in other industries, such as PLM and ERP are successfully applied to the software industry.

2.1.2 Lifecycle

2.1.2.1 Lifecycle definition

Software products go through a number of stages during their lifetime, such as conception, development and deployment. Each of these stages is associated with a number of activities. For example development includes among others design, implementation and testing. Each of these activities can in turn be subdivided into lower level activities and tasks.

This work defines lifecycle as *a set of activities and interrelations between them that are needed for the software product through its lifetime from conception to retirement*. While this definition seems trivial, it is important to explicitly specify it, because the term "lifecycle" is also used in other contexts. For example, Fuggetta defines it as: "a software lifecycle defines the principles and guidelines according to which these different stages have to be carried out" (Fuggetta, 2000). In his terminology a lifecycle can be waterfall, or incremental, or prototype based. In the context of ALM the meaning of lifecycle is different. ALM should support processes, but not impose

restrictions on the methodology of carrying them out. A central requirement to an ALME is to be flexible enough to support different development methodologies and guidelines. It should be customizable and not force customers into a predefined order of stages. Building a lifecycle support environment around fixed and rigid process will greatly limit its practicality. Every organization has a process that is somewhat different and process rigidity will increase the integration effort, which is already very high, as will be shown below.

2.1.2.2 Processes and activities

The ISO 12207 standard (International Standard Organization (ISO/IEC), 1995), (Singh, 1998) defines and discusses in depth software lifecycle processes. It establishes a top-level architecture of the life cycle. The building blocks of this architecture are processes and the relationships between them. The life cycle processes are divided into three classes:

1. **Primary**. They are the key pieces of the life cycle.
2. **Supporting**. They support other processes in performing some specialized function.
3. **Organizational**.

This classification is shown in Figure 1. The processes are further subdivided into their activities which in turn are built from their constituent tasks. Not all these processes are of an equal interest to ALM. It focuses mostly on the primary processes and many of the supporting processes, while leaving the organizational ones out of scope. Below is the discussion of the processes and activities that ALM deals with.

1. **Acquisition process.** It defines the activities and tasks of the acquirer of the software product or service. Usually these issues are considered out of scope of ALM and belong more to the ERP world.
2. **Supply Process.** It defines the activities and tasks of the supplier of the software or service. It begins from answering the acquirer's request for proposal, continues with the planning and execution of plans through the delivery of the service. Similarly to the Acquisition process the Supply Process is usually considered to be out of the ALM scope.

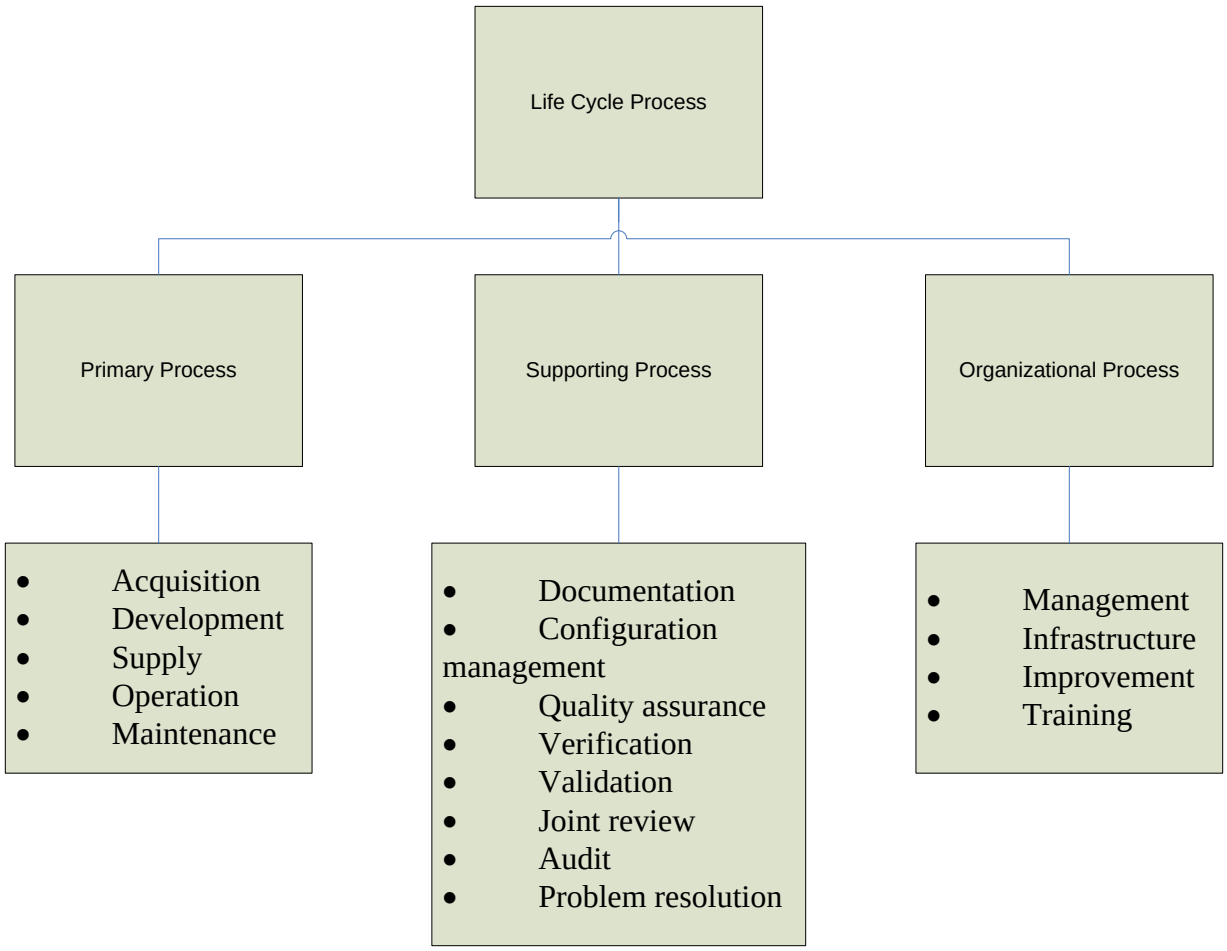

Figure 1. The Lifecycle Processes

3. **Development Process.** It contains activities and tasks of the developer of software and refers both to development of new software and modification of an existing software. The standard defines the following activities of the Development Process:

- Process implementation
- System requirements analysis
- System design
- Software requirements analysis
- Software architectural design
- Software detailed design
- Software coding and testing
- Software integration
- Software qualification testing
- System integration
- System qualification testing
- Software installation
- Software acceptance support.

The order of the activities in the list does not imply the order of their execution. The activities and their tasks can be used in any development model: the Waterfall, evolutionary, incremental and others as well as in a combination of these models. The Development Process is the heart of the software process and traditionally was the one that got the most attention. Its activities and tasks were the first one supported by different CASE (Computer Aided Software Engineering) tools and environments (Forte, 1991), (Fuggetta, 1993). Most ALM environments support many activities of the Development Process and many of them are focusing only on this process.

4. **Operation Process.** It contains activities and tasks of the operator of a software system and includes process implementation, operational testing, system operation and user support. This process is supported by the ALMEs to a lesser degree while support for the user support activity is more common than others.

5. **Maintenance Process.** It contains the activities and the tasks of the maintainer performed when a system is modified. The reasons for the modification can be either errors, problems, or system improvement. This process ensures that the modification, including system documentation, is

done properly. The standard defines the following activities of this process: Process implementation; Problem and modification analysis; Modification implementation; Maintenance review/acceptance; Migration; and Software retirement. Adding the Modification Process' support to an ALM environment can have a lot of value and a number of environments are already providing this support.

The supporting processes contribute to the quality and success of the project by supporting other primary or supporting process.

1. **Documentation Process.** The process defines the activities, which plan, design, develop, edit, distribute and maintain those documents needed by all concerned such as managers, engineers and users of the system (Singh, 1998). Adding support for the documentation process can give to an ALM environment a lot of value and a number of ALME`s (such as DiME (Koenig, 2003) and Orcanos (Orcanos)) are providing it. A most basic form of the documentation process support is simple storage of the documents' versions. A more sophisticated functionality is to keep track of changes to these documents. One of the most advanced features of such support is automatic documentation generation. It can provide a major improvement to the documentation ability of organization.
2. **Configuration Management Process.** This process is employed to identify, define, and baseline software items in a system; to control modifications and releases of the items; to record and report the status of the items and modification requests; to ensure the completeness and correctness of the items; and to control storage, handling and delivery of the items (Singh, 1998). A typical example of such software items are source code files. The Configuration Management Process is widely supported by ALM environments and it was one of the first processes that were facilitated by CASE systems.
3. **Quality Assurance Process**. Provides means for the acquirer to assure the conformance of products and services with their requirements. ALM environments can contribute to this goal by establishing and facilitating the right software process.
4. **Verification Process**. This process allows evaluating products or services of other activities for correctness and completeness. The verification can be applied to process, requirements, design, code, integration, and documentation.
5. **Validation Process.** It tests whether the final system fulfills its intended use. The ALM environments provide significant support for two of the quality control processes: verification and

validation. The supported activities are: test planning and execution, test plan coverage, defect tracking and reporting and others.

Other supporting processes: Joint Review, Audit, Problem Resolution Process are more procedural than technical are usually out of the ALMEs' support. However ALMEs can facilitate these processes by establishing effective communication means.

2.1.3 Management

In the previous sections we have described the software applications and the processes and activities that build the application lifecycle. These processes consist of numerous steps, operate on large volumes of data and involve numerous interactions. The software process must be managed in order to be effective. The processes should be guided; the data should be stored in a way that facilitates its accessibility, consistency and redundancy; the activities should be controlled and automated whenever possible. The technology that helps achieving these goals is known as CASE (computer-aided software engineering) – software tool support for the software engineering process (Sommerville, 2004).

The support that CASE tools and environments provide differs greatly and it depends on the functionality the tool or environment provides and the processes it supports. In general, these tools allow producing, storing, accessing, processing and searching different software process related data; they automate software process activities and tasks and provide guidance and facilitation for the processes themselves.

This brings us to the point of understanding of what exactly ALM is. Unfortunately, today there is no single agreed definition of this term. The author reviewed a number of existing wordings and created a unified definition of them. For each definition its main attributes were identified: category, scope, goals, means, managed entities. **Category** defines where ALM belongs to: is it a product category, activity, technology or a tool. **Scope** of ALM is the part of the supported software process. **Goals** are what ALM tries to achieve. **Means** are the activities that lead ALM to the goals. **Managed entities** are controlled and processed during the ALM activities.

Carey Schwaber from Forrester defines ALM (Schwaber, 2006) as "*The coordination of development life-cycle activities, including requirements, modeling, development, build, and testing, through: 1)enforcement of processes that span these activities; 2)management of relationships between development artifacts used or produced by these activities; and 3)reporting on progress of the development effort as a whole.*" Forrester's definition possesses the following attributes:

- **Category**: both product category and discipline. In principle, it is possible to accomplish the ALM goals without any tools.
- **Scope**: only part of the Development process from requirements analysis through testing.
- **Goals:** coordination of software development activities. In Forrester's definition, the ALM does not support specific activities, such as requirements analysis, but helps to coordinate between them.
- **Means:** the three main ways of achieving the ALM goals for Forrester are process enforcement, relationships management and reporting.
- **Managed entities**: process and artifacts relationships. ALM allows process defining and supports enforcement process execution and synchronizes artifacts produced by software activities.

Microsoft's definition (Microsoft, 2007) of ALM is: "*ALM describes methods to manage software development and IT initiatives by automating the process from end to end, and integrating the information from the various steps…A key characteristic of ALM is that all the project stakeholders (from the business and IT functions) share the same pool of up-to-date information. This includes business sponsors, users, project managers, architects, developers, testers, and system administrators. The typical activities supported by ALM include requirements gathering, solutions modeling, visual design, coding, testing, deployment, and issue tracking. ALM tools link together the artifacts that result from these activities.*" This definition is quite close to the Forrester's one and its attributes are:

- **Category**: methodology, not product.
- **Scope**: supports the whole Development, Operation and Maintenance processes with a focus on IT and business needs.

- **Goals: "***manage software development and IT initiatives*".
- **Means:** process automation, information integration and centralized management, reporting.
- **Managed entities**: process and artifacts relationships. The artifacts themselves are not necessarily managed by the ALM tools in Microsoft's vision. This is the same as in the Forrester's definition.

Borland's definition (Borland, 2007) of ALM is somewhat different. It defines ALM as a market category populated with suites of products. These products "*address the challenge of increasing the consistency and predictability of software delivery*" and are team-based development platforms. The Borland's definition's attributes are:

- **Category**: market category and tools.
- **Scope**: supports the Development process, supplies data for project management.
- **Goals:** consistent and predictable software delivery.
- **Means:** measurability (quality, progress and risk management and reporting), alignment (business priorities management, requires traceability and data integration), discipline (software process management and automation).
- **Managed entities**: while not stated explicitly, it seems that ALM platform in Borland's definition manages the same entities as in other definitions. In addition, Borland stresses the metrics dimension of data processing.

Finally, IBM defines ALM is a "*market need for a suite of integrated tools to help teams manage all of the assets in a software development project*" (IBM, 2008). The IBM's definition attributes are:

- **Category**: integrated tools.
- **Scope**: from the project inception and portfolio management, though development to operation and maintenance.
- **Goals:** "*streamlining a team's ability to produce a release of software*".
- **Means:** data integration, process governance, team cooperation and integration support.
- **Managed entities:** not specified.

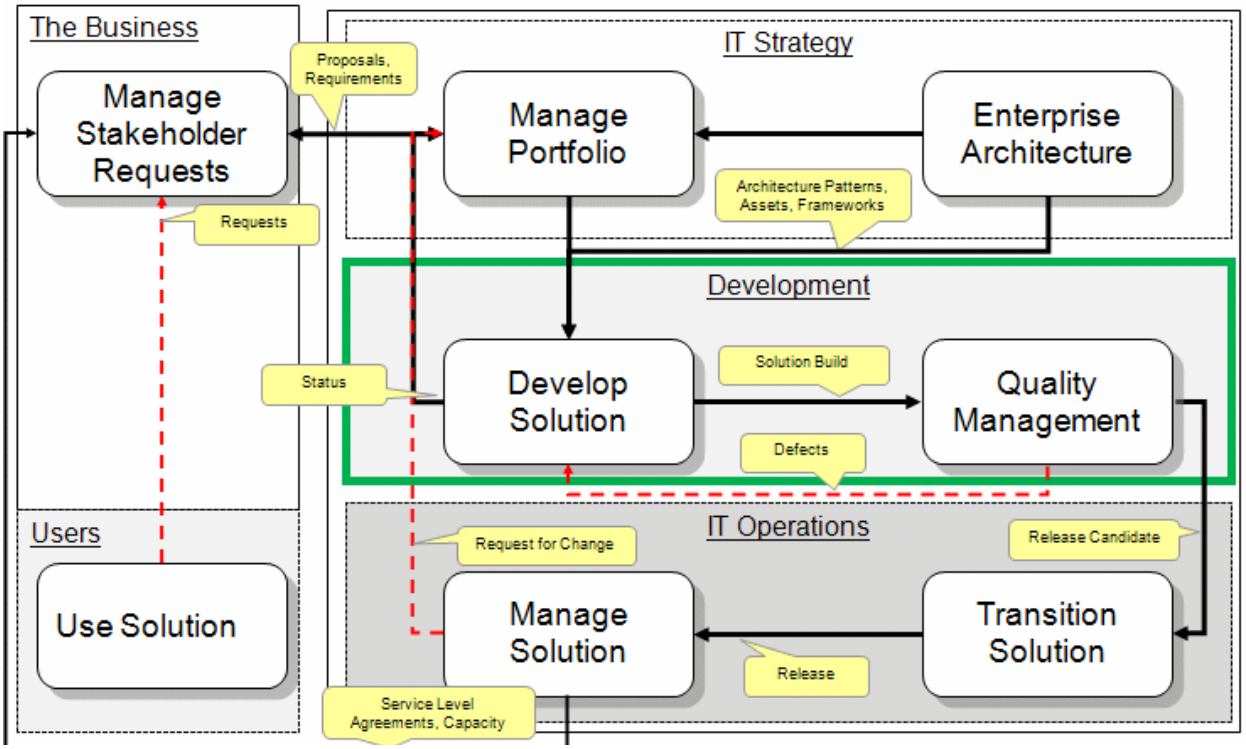

Figure 2. The scope of ALM (adopted from (IBM, 2008))

After reviewing a number of existing ALM definitions it seems that despite the wording difference there is a shared common concept of what ALM is. The ALM unified definition attributes are:

- **Category**: methodology and integrated tools supporting it.
- **Scope:** full application lifecycle, from inception through operation.
- **Goals:** making software delivery more efficient and predictable.
- **Means:** coordination and integration of data and artifacts produced by different software development activities, defining, automating and supporting the processes and creating visibility into the process state for all stakeholders.
- **Managed entities:** process and artifacts relationships.

In addition ALM should provide answers to business needs and facilitate distributed and multiplatform development by geographically dispersed teams.

2.1.4 Environments

ALM environments represent a category of CASE (computer-aided software engineering) products. A large number of CASE products exist and they vary in many characteristics. In order to facilitate understanding of CASE tools, their functionality and relationship to each other and their role in the software process, one needs a classification. There are several ways to classify CASE tools; each one of them looks at CASE tools from a different perspective.

Sommerville (Sommerville, 2004) proposes to focus on three perspectives that build together a complete picture of CASE tools. These perspectives are:

1. **A functional perspective**. CASE tools are classified by the functionality they provide. The functional classification divides the tools into planning, editing, configuration management, change management, prototyping, language-processing, method-support, program analysis, testing, debugging, documentation and other tools.
2. **A process perspective**. CASE tools are classified by process activities they support. The tools are grouped into categories: specification, design, implementation, verification and validation. The same functional tools can support more than one process and therefore belong to many categories. For example, tools for text editing may be used throughout the software process.
3. **An integration perspective**. CASE tools are classified by how they are organized into integrated units. Wasserman (Wasserman, 1990) identified five kinds of integration:
    a. **Platform,** which is supported by framework services.
    b. **Presentation,** which is concerned with user interaction.
    c. **Data,** concerned with data consistency regardless of how it is operated by tools.
    d. **Control,** concerned with tools interoperability and communication.
    e. **Process,** concerned with effective tools cooperation in support of software process.

Fuggetta (Fuggetta, 1993) adds another dimension to the classification, which is a higher level view on CASE products. He classifies the CASE systems by the breadth of support for the software process they provide.

- **Tools** support only specific tasks in the software process such as code compilation, spelling check. They may be general purpose (e.g., word processor) or integrated into workbenches.
- **Workbenches** support one or a few process activities such as specification, design. They are sets of tools with some degree of integration. Very common are programming workbenches that includes different tools that support programming: a text editor, a compiler, a linker, a debugger.
- **Environments** support all or at least a substantial part of the software process. Usually they include a number of integrated workbenches and tools.

In practice, the boundaries between theses classes are not so distinct. Tools may have built in support for different activities. It may be sometimes hard to classify a specific CASE product into one of these three categories. Nevertheless, the classification is still useful to assess the extent of the process the product supports.

ALM systems belong to the third category in Fuggetta's classification. They are integration of workbenches that support different activities and processes in the IEEE standard terminology. When building an ALM environments' classification we will address all these perspectives: we will inspect the functionality they provide, the parts of the software process they support, the quality and kind of integration that exists between the parts of the environment.

2.2 History

2.2.1 Evolution of CASE

CASE tools evolution is driven by a number of factors.

- **Technology** influences it in two aspects: what CASE tools can do, and what they are required to do. CASE tools role is to support the software process, so that the requirements for them are dictated by the applications they need to support, and these depend on technology progress. On the other hand, the technology defines what part of the requirements CASE tools can implement. Sometimes the technology is not mature enough to satisfy some existing need.

- **Software development methods** are another factor of CASE tools progress. As development methods progress they set new challenges to the support CASE tools need to provide.
- **Economical value** should be taken into account as well. Software development is a business and it gets more focus in the areas that deliver more value to the software systems vendors.
- **Incremental evolution** - CASE tools develop **incrementally**, from simple to more complex, getting into new areas of the software process after reaching needed maturity in the existing ones.

First-generation CASE tools appeared in the 1970's. Most business applications then were batch transaction-processing systems. The first CASE tools were generally mainframe and text based (Norman & Chen, 1992). There were also compilers and other programming supporting tools. The development methods were mostly structured programming and structured design. These methods required support for automation of gathering and analysis of the information they produced. With the advent of graphical user interfaces integrated programming environments that gave better support to structured design and analysis appeared. Among other features they offered dataflow diagrams and structure charts creation and editing.

In the 80's object oriented methods were developed to overcome the deficiencies of the structured methods. More complex systems were created using them and they required more from CASE tools. The next generation provided collections of tools that supported more activities, like planning, analysis, design, programming, testing. Appearance of code generators and fourth-generation languages have simplified activities related to coding itself so the development bottleneck has shifted to requirements engineering, system planning and other so-called "upstream activities". Big volumes of information these tools operated required storing this data in repositories. The first repositories were mere project dictionaries, but as technology advanced and the system became larger the repositories grew to offer enterprise-wide support. However they were limited to tools from the same vendor and to a certain methods of application development.

In the 80's and beginning of the 90's there was an expectation that the next step in CASE evolution are integrated CASE environments that allow the management of activities through the whole software process, including integration of tools developed by different vendors. A number of initiatives focused on creating frameworks for such integrations emerged. The most important of these initiatives, IPSE (Integrated Programming Support Environments) and PCTE (Portable Common Tools Environment) got a lot of attention, were massively supported by different software engineering institutions and discussed at many conferences. Ten years later it turned out that all these initiatives are virtually dead. Most software vendors were developing standalone tools and toolsets and investing massive efforts to achieve some integration between them.

The reasons for this failure to achieve a common integration platform are still to be studied. A number of factors led to it. It seems that **technology** was not mature enough to provide such large scale integrations. The proposed frameworks were based on the technologies (C language interfaces, Ada) that were getting obsolete and inappropriate for large systems integrations. The repositories were not mature enough to provide higher levels of abstractions required for centralized management of activities from wide range of lifecycle activities. **Process immaturity** is another reason for this failure in my opinion. Providing only a technological platform for integration is not enough for developing useful integrated systems. The integration needs to be much deeper then just the ability to transfer bytes in a common format between the tools. This integration can be reached when the software process is understood enough and the tools supporting different phases of it are mature themselves. The need for even faster development that we have seen in 90's resulted in new methods of software development, more agile, more dependent on COTS (commercial off the shelf) components, more globally distributed. These methods required software community and vendors to focus on creating proper tools and environments for downstream software activities. These tools brought more value to the software developers and were necessary building blocks for the development lifecycle. So the questions of integration were postponed until the workbenches supporting individual processes are mature enough.

It seems that now the software development world has reached the state of readiness for integrated application lifecycle environments. The necessary components are present. The existing development processes are understood well enough and get the necessary support from tools ceasing from being a bottleneck. The wide spread of SOA (service oriented architecture) creates a proper platform for the integration – products are exposing their functionality in a form of Web service interfaces. The database technologies have progressed a lot and can give the necessary robustness, reliability, level of abstraction and power to manage large volumes of data coming from different stages of the life cycle. Finally, there is a growing business need for an effective and integrated solution that will manage the whole application life cycle. Software development has become a large and important industry sector and to succeed in this business executive management requires a holistic approach similar to what ERP provides for other enterprise resources. The author thinks that all these factors together have sparked a new interest in ALM which we experience today.

2.2.2 IPSE

A process for establishing requirements for the integrated software engineering environments started in 1979 and as a result of it the "Stoneman" report (Buxton, 1980) was published in 1980. While focused on Ada Programming Support Environments (APSE), this document contained ideas that were not Ada-specific and became a basis for many subsequent works on IPSE`s. This report contained a vision of an integrated suite of tools supporting the whole lifecycle. The architecture provided platform independence of user programs and software tools, interoperability between independently developed tools and user interfaces built above old-fashioned command line styles. The core idea of the architecture was a layered approach where lower levels provided services to higher ones.

- **Level 0** contained hardware and operating system
- **Level 1** contained a kernel presenting a platform independent interface to its services that included database, communications and run-time support functions.
- **Level 2** contained a minimal set of tools supported by the kernel which is necessary and sufficient for development and support. The tools included: editor, compiler, debugger, configuration management tools and others.

- **Level 3** contained environments (APSE`s) that extended the minimal set to support particular methodologies or applications.

This architecture inspired a number of efforts to build such integrated frameworks. Through the next ten years until the early 90's a large number of such efforts were undertaken both in Europe and the US. Some of these efforts were heavily funded and lasted for many years. In the US the most important such projects were APSE projects sponsored by the US Army and Navy at a total cost of approximately 100 million dollars and software engineering environments (SEE`s) work funded by the US government estimated at 75 million dollars. In Europe, an Alvey project (Morgan, 1987) included a number of IPSE initiatives with total investment above 20 million pounds and the Eureka Software Factory project received a total of about 400 million dollars (Brown, 1993). Despite these investments and thousands of man-years put into these projects after 10 years they failed to achieve their goal – widely accepted and implemented integrated environments supporting all aspects of software development. More than that, even in the organization that sponsored some of these projects (like DoD) they did not succeed to materialize this vision.

Brown (Brown, 1993) gives a detailed and deep analysis of the achievements of IPSE`s and the reasons for their failure. He lists 2 main contributions of the work on IPSE`s. The first one is their role in the database technology revolution that took place in the 90's. Initially IPSE projects tried to utilize existing commercial database systems for their needs. However, it turned out that they were not suitable for managing the large volumes of complex and interrelated data that IPSE`s required. This led to massive research in the database world to support these IPSE's requirements. This research activity resulted in breakthrough in such areas as Object Oriented Databases and catalyzed the progress in databases in general. The second major impact of IPSE`s effort was on the software development process. In trying to support the software process as a whole, the IPSE's projects had significantly improved the understanding of what the software process is, how to define, model, automate and control it.

Brown also names the reasons that caused, according to him, the lack of practical success of IPSE`s. He mentions four important causes of a very low adoption rate of IPSE`s in commercial organizations.

1. **Cost.** IPSE`s are trying to solve very large and complex problems and thus are very large and complex systems themselves. They often require dedicated and costly hardware, a long and intensive implementation phase, massive employee training, hiring dedicated IPSE support staff and high maintenance fees. Multiple tools comprising the IPSE usually are updated periodically causing additional efforts to cope with the changes. The organizational price is very high as well. In order to adopt the IPSE successfully the organization very often needs to change the way it works and sometimes the organizational structure, adding new positions needed to manage the changed process. All these factors make shifting to IPSE-based process a risky endeavor that will result in at least some initial decrease in productivity and higher operational expenses. In addition, due to their size, complexity and lack of maturity the IPSE's systems suffered from low stability, slow performance and functional problems, which only added to the reluctance to their adoption.
2. **Lack of flexibility.** Despite the declared goal of providing an open infrastructure that supports easy integration and cooperation of independently developed tools, the real life IPSE`s were a long way from that ideal. More commonly they were developed with a fixed set of tools which allowed extension by referring to a small number of vendors. They also were suited mostly for some specific type of development process. This made them less appropriate for organization with a diverse range of projects using a number of different tools coming from multiple sources.
3. **Lack of standards.** Quite ironically, IPSE`s, which should provide a common platform for integration, in practice did not have a common understanding of what that platform should be. Long debates took place about what parts of IPSE technology should be standardized and what these standards should look like. New techniques and approaches were constantly developed and compared. This situation is not attractive for commercial organizations that require portability, compatibility and extended product life from a long term strategic investment, which IPSE`s are.

4. **Little evidence of practical success.** Organizations look for practical evidences that IPSE`s really deliver what they promise. Unfortunately it turned out that the available successful implementation data was quite hard to analyze. Even from the success stories it was not clear that the IPSE`s were the factor that contributed the most to the productivity gain or they were just part of a larger software process improvement.

Brown also reviews the basic mistakes that took the IPSE`s into the wrong direction. They include the following issues.

1. **Focusing on the technical issues instead of user needs.** The original work on IPSE technology put the main effort on defining and solving purely technical issues of large and complex data sets management and creating generic open interfaces. At the same time very little attention was put into leveraging this infrastructure to satisfy user needs. This led to extremely large systems that provided no specific support for the functionality that the user actually wanted.
2. **Superficial understanding of tool integration.** This issue is reviewed in depth in an article by Brown and McDermid "Learning from IPSE's mistakes" (Brown & McDermid, 1992). The authors explain that in orders to reach the IPSE`s goals the technology should address the following aspects of integration:
    a. **Interface integration.** Different tools should provide consistent user interaction experience. This eliminates the need for user training and "context switch" during using different IPSE tools.
    b. **Process integration.** The environment should support consistent software development methodology though the whole life cycle. This goes together with a requirement to allow user to define the specific methodology to use, or at least allow for this configuration and customization.
    c. **Tool integration.** In Brown and McDermid terminology this refers only to the ability of separate tools to share data.
    d. **Team integration.** The environment should provide effective means for teamwork, which includes team members' communication, methods for work sharing and separation.

e. **Management integration.** The environment should help managers to stay updated and to control development.

This classification is quite close to five types of tool integration defined by Wasserman (Wasserman, 1990). Brown and McDermid claim that most IPSE`s and especially IPSE infrastructures focus exclusively on tool integration and even there the state of technology is unsatisfactory. This is partially explained by the fact that at the time (in the early 90's) the technology for group work facilitation and process enablement was immature. But another not less important reason for this state of integration is an overly mechanistic approach to it.

Tool integration has several levels that present different degrees of integration provided. The lowest one, "carrier", allows tools to pass raw binary data. There is no understanding what is inside this data. The next level is "lexical" – when tools share lexical conventions about the data so they can recognize the parts that comprise the data stream and operate on the parts they support. The next "syntactic" level is sharing a set of data structures. An example of such integration level is database schema. At the next, "semantic" level tools agree on the meaning of the data structure elements and operations on them. This semantic integration level is the one that can allow automating of many development tasks. The highest level is "method" integration. It actually goes beyond the tool integration in their definition and reaches the process integration. At this level tools share the notion of development process and know each tool's role in it. The tools interact notifying each other about the operations performed to create a single consistent process flow. The analysis of the existing IPSE`s revealed that they provided at most the syntactic level of integration, while the real productivity gain and the functionality needed for user application starts from the semantic integration level. Without them the practical value of the integrated infrastructures is questionable.

3. **Misunderstanding of the software process.** IPSE's goal is to support the development process. They can't replace it. If the organization does not have the right development process in place trying to automate it will not bring any real value. On the other hand, an organization lacking the right development process rarely feels a need to automate it at

all. As Humphrey's (Humphrey, 1989) research has shown, most software organizations in the 80's had a particularly immature software process. This means that both the organizations were not ready for implementing an IPSE solution and the IPSE solutions that existed had little understanding of the process they should support.

4. **Overlooking the organizational and business aspects.** A very large part of IPSE`s projects completely ignored these aspects of IPSE adoption. As explained above, IPSE system implementation incurs a large price on the business.

A different strategy could bring much bigger success to IPSE`s. This strategy should include:

a. **Smaller projects** that attack specific user problems, based on real world user requirements and development processes
b. **Evolutionary implementation.** Allowing adopting the of the IPSE part by part, trying to focus on practical issues instead of generic, but not practical solutions
c. **Targeting real world development platforms.** A lot of IPSE research was done for Ada environment, even though it has only a small fraction of the development market. At the same time many popular operatinng systems and environments got little attention**.**

To summarize, the failure of the IPSE research to reach wide acceptance and practical results is rooted in the gap between the researchers and the users. Concentrating the IPSE research on the technological issues led to lack of attention in making this technology useful to the development world. Trying to solve a very big and complex problem without properly understanding its context led to an impractical solution that could not solve even a fraction of it. Learning to crawl before trying to walk could bring the IPSEs to its goal.

2.2.3 PCTE

The acronym PCTE stands for Portable Common Tool Environment. A project to produce such an environment was started in 1983 in Europe and its aim was to produce **an interface specification** that would allow tools integration. The first such specification was published in 1986 (Ada Joint Program Office, United States Department of Defense, 1986). This specification

was improved and developed over the years both in Europe and the US and finally resulted in an international standard produced by the European Computers Manufacturing Association (ECMA) in 1990 (European Computer Manufacturers Association, 1990). This standard was called ECMA PCTE and was an abstract interface specification. Language bindings of this specification to C and Ada were produced in 1991. It is very important to remember that PCTE defines the interfaces, not their implementation or the complete environment populated by tools. In fact, it does not devote any attention to the specific tool requirements and the issues of their coordination and cooperation.

The PCTE defines interfaces necessary for data management, process control, security, network management, auditing and accounting (Long & Morris, 1993). The PCTE data management system is called Object Management System (OMS). Every data item in OMS is *an object;* an object may have content, similarly to a file in an operation system. Objects may be either *primitive* or *composite*, have *attributes* and *type*. Objects can be *linked* to other objects and links are also typed and have attributes. The PCTE specifies ways to manipulate objects, links and attributes. Types are defined and managed in Schema Definition Sets (SDS). SDS`s are stored in the OMS and can be defined and manipulated using the PCTE interfaces. Despite its terminology, the PCTE data model is an entity-relationship one and not object-oriented.

Another important aspect of the PCTE is process management. The PCTE specifies advanced process management facilities allowing for start, stop, pause and resume processes as well as query the process status. Processes are also objects in OMS. Processes can communicate using pipes and message queues. PCTE specifies methods of performing atomic transaction activities by processes so that the data integrity is achieved.

PCTE defines a sophisticated security model to satisfy the needs of defense systems. Both discretionary and mandatory access controls are provided. PCTE supports a notification mechanism allowing a process to be informed when some change occurs to an object in the OMS. PCTE was designed for a network of workstations and specifies sophisticated networking functions including data and schema replication.

Despite a lot of efforts invested in PCTE and its standardization the practical success of PCTE was quite limited. The most widely distributed PCTE implementation was Emeraude V12 that was available on a number of Unix platforms. This system implemented an older PCTE 1.5 standard and has not released the product fully supporting the ECMA standard. A number of commercial tools for this platform were created, but it seems that by the mid 90's these activities had stopped without reaching any significant practical results.

The reasons for the PCTE failure are to be discussed. A recent work by Lewis et al. (Lewis, Morris, Simanta, & Wrage, 2008) calls PCTE an "irrelevant standard" and claims that it failed because too few software vendors supported it. While this is true, in order to learn from the PCTE mistakes we need to look deeper and understand why the number of vendors was small. In my opinion the very basic problem with PCTE was its impracticality. The efforts invested into developing the standard were mostly focused on a theoretical question of providing a comprehensive interface specification, mostly in some basic infrastructural areas (data management, process management, security). While resulting in some good level of understanding in these areas, this gave no insight in specifying the needed functionality for building useful tools based on this infrastructure. The provided specification did not cope with important integration aspects that are vital for a successful implementation. In general, it seemed that the PCTE efforts tried to solve a problem that was rather too big at the time. It was too early - technology-wise, process-wise and business-wise. As a result, the PCTE technology ended up in defining solutions for a very basic parts of the overall life cycle support. The same issues were addressed by other technologies, like object-oriented databases and operating systems, so the relative advantage that the PCTE standard offered was not very significant. PCTE also could not succeed because of the problematic implementation model, or rather absence of a practical one. Implementing a PCTE solution posed a number of very challenging questions that had no good answers. First, there had to be a reliable and stable PCTE framework implementation. Because of the complexity and the instability of the specification, developing such a framework was a large development project. Because its commercial value was unproven, not many development organizations were going to invest in developing such a framework. PCTE required from tool

vendors major efforts to conform to its interfaces. Migrating existing tools to the PCTE framework was also a tedious task and not always possible without tool rewriting. This is not to mention that there was a serious basic design flaw in the PCTE specification that made the PCTE systems impractical in activities that required fast retrieval of data. Finally, the PCTE gave no guidance on process integration and building an integrated software engineering environment. It did not provide any specific support for the functionality required by the CASE tools and did not define how to integrate different tools using its interfaces.

To summarize, the PCTE effort resulted in a better understanding of the requirements for services needed for building an integrated software engineering environment. However the PCTE specification did not address a number of crucial practical issues which caused a very little market penetration of products based on the PCTE standard and eventual desertion of this direction. Nowadays a lot of problems solved by the PCTE are addressed by other technologies: databases, operating systems and development frameworks. One still can learn from the PCTE's achievements and mistakes and thus build better lifecycle support.

2.2.4 ALM 1.0

The term ALM 1.0 was introduced by Forrester analyst Carey Schwaber (Schwaber, 2006). It refers to the attempts to reach the ALM goals as identified above by adopting the existing tools and environments. This is not an easy task because most of these tools are functional silos that support one kind of activity and were not built with tight integration in mind. Because of that the ALM 1.0 approach has a number of inherent problems.

1. **Fragile and superficial integrations.** In order to tie together existing tools, organizations establish tool-to-tool integrations using the APIs the tools expose and repository synchronization mechanisms. However, in practice these integrations are hard to set up, very difficult to maintain and they are expensive to upgrade when one of the tools changes. Because of their point-to-point nature $O(n^2)$ integrations are required for n tools. And, at the end, these integrations still are not deep and reliable enough to meet business needs (Schwaber, 2006), (Shaw, 2007).

2. **Traceability and reporting problems.** In order to create a project status report it is necessary to collect and process data from a number of tools, each focusing on a specific discipline: project management, test management, defect management, requirements management. Each tool keeps this data in its internal format, which makes data coordination and consolidation a hard problem (Shaw, 2007).
3. **Redundant and inconsistent feature implementation.** Each tool addresses common problems such as team members' cooperation and collaboration, reporting, security in addition to the core functionality it provides. However, each tool achieves this in a unique manner, which leaves practitioners with the same functionality provided by many tools on an inconsistent basis. Another common practice by tool vendors is to add features from adjacent disciplines, e.g., a test management tool vendor that decides to add defect management to its product. The purpose of such addition is to increase the tool's market share, but it leads to the products stuffed with many features that are less usable but require more integrations.

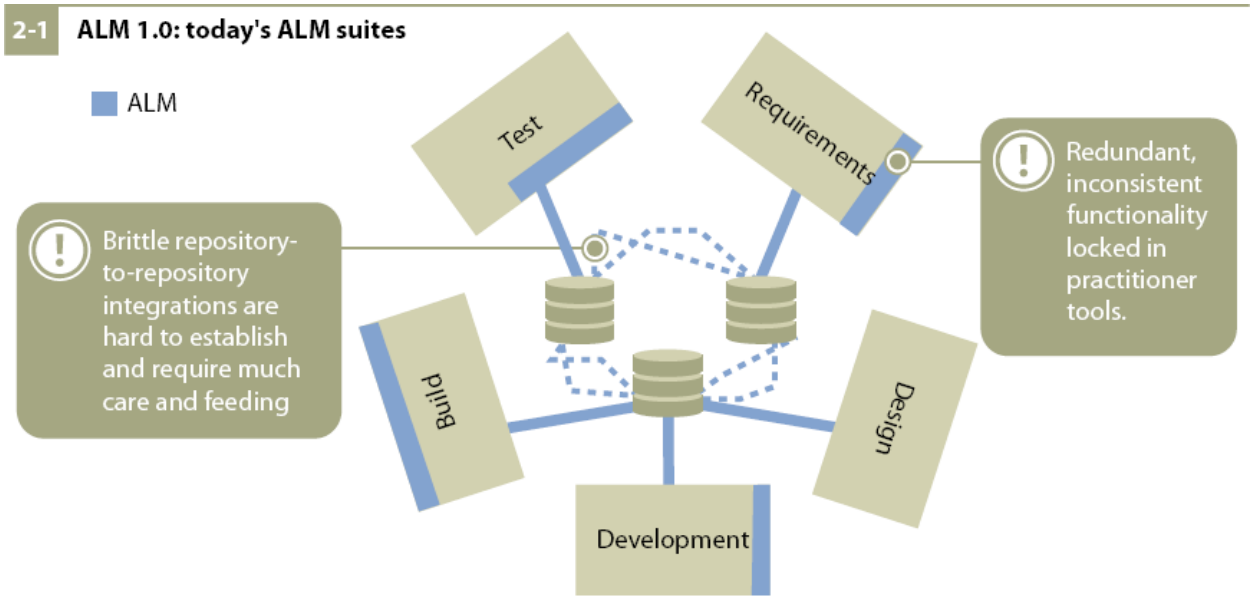

Figure 3. ALM 1.0 suites (adopted from (Schwaber, 2006))

Thus the ALM 1.0 approach provides only a very partial answer to the needs of the today's application lifecycle management. ALM 2.0 environments' goal is to provide a comprehensive, robust and scalable solution to these needs.

3 ALME Classification Framework

This section presents a classification framework developed by the author. The framework will be used for a comparative study of a number of existing ALM environments. The goal of this framework is to provide a tool for assessing ALM environments, understanding their characteristics, and discussing and comparing them.

3.1 Classification Aims and Principles

The framework should take into account a number of factors. First, it should provide us with a clear understanding of how the ALME being studied serves the ALM goals, as identified above: integration, process automation and reporting. In the case when a specific environment explicitly sets goals that differ from the ones we have defined as common, the framework should capture that.

The integration aspect should be covered in depth. The framework should consider both different *kinds of integration* (platform, presentation, data, control, process) as defined by Wasserman (Wasserman, 1990) and *integration level*, as defined by Brown and McDermid (Brown & McDermid, 1992).

Also, the framework should reflect the breadth of support the ALME provides for the lifecycle processes. The existing ALME`s target many, but not all lifecycle processes, and many of them deliberately exclude some activities (e.g., portfolio management) from their scope.

The framework should consider the technical aspects of the ALME implementation: the supported platforms and languages, data management and integration strategy, networking and the distributed work support it provides, extensibility and architectural patterns it utilizes.

As shown above, the technical factors are not the only ones influencing the practical success of the integrated system. The framework should capture the organizational and process aspects of ALMEs and the support they offer in meeting such needs.

Wherever possible, we should include available experiences from the studied system field usage. This will allow assessing the differences between the advertised and actual capabilities, strengths, deficiencies and quality of the product. Also practical experience will help us identify new aspects to consider when studying (and developing) ALM environments, aspects we otherwise may not foresee.

3.2 Related work

To the best of the author's knowledge no classification framework for ALMEs currently exists. However, there are two main groups of sources that can help build one. On the one hand, a number of classification frameworks for software engineering environments exist, including the ones for the integrated Software Engineering Environments (SEE`s). The most fundamental of them is a "Reference Model for Frameworks for Software Engineering Environments" (SEE RM) (NIST/ECMA, 1993) produced jointly by NIST and ECMA. This work focuses on the SEE frameworks, which are a part of SEE architecture along with tools. The difference between them is that tools support the life cycle processes while the framework provides infrastructure capabilities for the tools. The SEE frameworks simplify the tools' construction and provide facilities for integration between tools and for tools' platform independence. The SEE RM sees the SEE framework as a provider of a set of services. These services comprise the following service groups:

- **Object Management Services** deal with all aspects of data management including definition, storage, management and access of object entities and relationships between them.
- **Process Management Services** support definition, enactment, management and governance of lifecycle processes**.**
- **Communication Services** support other services communication, such as data sharing, messaging and notifications.
- **Operating System Services** provide platform independent access to the services usually exposed by the operating system.

- **User Interface Services** deal with all aspects of interaction between user and tools.
- **Policy Enforcement Services** manage security and integrity.
- **Framework Administration Services** include means for incorporating new tools into an environment, resource and license management.

These groups are somewhat parallel to Wasserman's abovementioned classification (Wasserman, 1990).

The CASE tools classification frameworks, such as the ones by Fuggetta (Fuggetta, 1993) and Sommerville (Sommerville, 2004) are another point of reference for building our ALME classification framework. Another detailed tools classification is the one by Oliver (Oliver, 1994). This work proposes ten independent classifications, each one focusing on some aspect of the classified system, such as inter-operability, extensibility, use, platforms, methodology and others.

Another helpful resource in building the classifications are analysts' reports on the ALM market and tools. These reports (Schwaber, 2006), (Rotibi, 2006) provide a broad and deep review of the current ALM market, discuss business, technical and organizational aspects and offer a lot of useful insights on the issue.

3.3 Classification Framework

The purpose of this classification framework is to build a high-level broad picture of the studied environments. Therefore it will not cover the lowest level details (as SEE RM does, for example), but instead will give an overview of a wide spectrum of aspects. The framework will cover two categories of aspects: technical and non-technical. The non technical aspects will include the organizational and the business ones. This separation is sometimes not sharp, as technical characteristics can have significant organizational and business impact. An example of such tight connection can be vendor and platform support. Still, categorizing characteristics according to the above distinction between technical and non-technical aspects adds to the clarity of the classification.

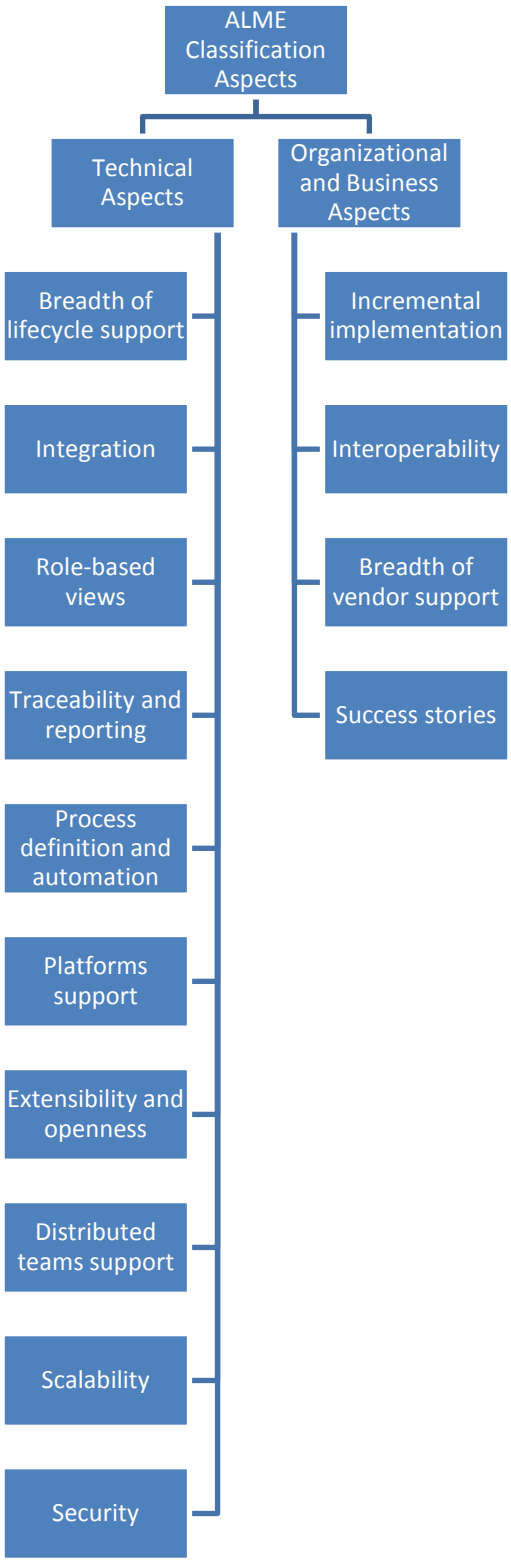

Figure 4. The developed ALME Classification Framework

3.3.1 Technical aspects

Breadth of lifecycle support. Analysts' market research shows (Rotibi, 2006) that most, if not all, existing ALM environments support only part of the full application life cycle. Some of them focus on development and pre-deployment processes and some – on post deployment and operations. Our framework will clearly identify the stages covered by the environment. Apart from the stages that the system already supports directly, the framework should also capture the stages the ALME is going to support in future or provide indirect support to (by interfacing with other systems).

Integration. Integration is one of the most important aspects of the ALME classification. As one of the goals of ALME is to provide a deep integration between tools and environments, our framework should describe in details the depth of each integration type (presentation, data, control, process). The framework will pay special attention to the data and process integration by identifying the level of integration (lexical, syntactic, semantic) that the ALME achieves. Data storing and synchronization strategy is one of the fundamental concepts of ALME architecture. Major competing strategies of data synchronization are: *single repository* and *repository-neutral* ALM platforms. The single repository approach (taken, for example, by IBM, Serena and MKS) provides deeper, more powerful and robust data integration, while repository neutrality promises greater interoperability, better cross-platform development support and, at least in theory, frees the implementing organization from the need to migrate existing data.

Role-based views. ALM ties together different parts of the organization and the ALME should provide each organization role with the view of the data that is relevant to its function. Specifically, the ALME should provide business users, architects, IT and operations staff with tools presenting consistent and up-to-date views of the data reflecting the product state. These views should come in addition to the views it must provide to the development, quality assurance and product management staff.

Traceability and reporting. These capabilities deliver a significant part of ALM systems' business value and therefore our framework should reflect them. At the same time, traceability

and reporting abilities heavily rely on the integration depth provided by the ALME and its infrastructure. For this reason, a lack of some specific reporting ability has less influence on the overall ALME quality, provided the ALME infrastructure allows for adding this feature.

Process definition and automation. Process automation is similar to reporting, in that it delivers a significant part of the ALM value to organization and builds upon its infrastructure's integration capabilities. Specifically, the process integration level is a basis for the ALME process automation. Still this aspect deserves its own place in the classification because of its importance to the ALM goals. Governance abilities offered by the ALME also belong to this category.

Platforms support. Today's software development is characterized by a very heterogeneous and diverse environment. Application development involves different development languages (e.g., C/C++/Java), development and deployment environments (e.g., .NET, J2EE) and different operating systems (e.g., Windows, UNIX, embedded OSes, mainframe OSes). Sometimes all these ingredients are part of the development process for the same application or application suite. The more platforms an ALME can support, the more significant is its impact on the organization.

Extensibility and openness. These characteristics are very important for the practical success of an ALME. A single ALM platform cannot address all the needs of today's highly diverse development reality, and surely can't address the needs of tomorrow. Openness and flexibility are critical for an organization that follows the best-of-breed approach when choosing its development tools and does not want to be locked into a single system or vendor. An ALME can follow different paths to achieve openness: rich service interface exposure, usage of standard data and interface formats (XML, web services), provision of connectors to other systems or open source code.

Distributed teams support. Geographically distributed development and operations are very common today. Usually an enterprise will have a number of development teams residing on

different continents and separated by many time zones taking part in the same development project. It will probably have also product management, business analysts and quality assurance teams spread all over the globe. In addition to that, today's technology brought new mobility opportunities with GPRS, 3G data services and a VPN access to the network. An ALM platform has to support such a distributed mode of work in an effective and seamless way.

Scalability. Every organization, regardless of its size, has its product lifecycle that has to be managed. An ability to support organizations of different sizes and natures, from small coherent teams to huge international enterprises is an important characteristic for a lifecycle management system.

Security. Application lifecycle deals with first-class business information and therefore has to conform to strict, reliable and adaptable security regulations.

3.3.2 Organizational and Business Aspects

The ALME's overall quality and its chance to succeed depend by and large on how well the ALME considers the organizational and business sides of ALM. This lesson must be learnt from the history of IPSEs' failure and from the success of ERP systems in today's industry. The author proposes to include in the framework the following characteristics to describe the non-technical facets of an ALME.

Incremental implementation. ALM systems naturally consist of a number of subsystems that support different functional and organizational areas. The ability to implement selected parts of an ALM system provides great advantage to an organization that needs only some of the functional capabilities the environment offers. For example, the organization may not be able or may not want to change parts of the existing solution. Or it may prefer implementing the system in a limited scope to assess its efficiency, quality and suitability to the organization's needs. A lack of this flexibility makes it very hard for an ALM system to gain a wide installation base. Organizations now understand the big price such installations incur and are not willing to pay for something that has not proven itself yet. Only the biggest vendors that supply complete lifecycle

support for the organization will be able to have these all-or-nothing ALME implementations installed.

Integration and interoperability with existing solutions and tools. Already mentioned in the technical aspects, this characteristic has a critical organizational and business impact. Every organization has some automated support for parts of the application lifecycle in place. Usually the existing systems hold a lot of data that is vital for the organization's business and are integrated with its existing development process. An ALME that can integrate with the existing tools has a good chance it will be tried and eventually used by the organization, while an environment demanding replacement of the existing solution will inevitably face serious opposition from inside the organization. Providing data migration answers this problem very partially, as it does not address the process change impact.

Breadth of vendor support. In general, organizations don't want to find themselves locked into a solution provided and backed by a single vendor or a small group of vendors. This limits greatly the organization's ability to choose the right tools for its needs and impacts its business freedom. Consequently, ALM environments that are based on standards with a wide vendor support base will have better chances to succeed. This rule can be less important for the largest vendors that usually provide the entire IT solution, but in this case the ALM support this solution provides will probably bear little influence in overall considerations regarding the choice of a specific vendor and its IT solution.

Success stories. This is a purely practical, but nevertheless an important characteristic. Before going into a long and expensive ALM system implementation phase, the organization looks for the experience of some other business that has gone through this process. This is a place for practical evidences for the system's real strengths and deficiencies, differences between the marketed and actual capabilities and overall product quality.

4 Case Studies of Current ALME Projects

In this chapter we will apply the developed ALME classification framework to a number of current products. Among existing ALM solutions we have studied leading commercial products: IBM's Jazz Platform and Team Concert and Microsoft Visual Studio Team System. The author has also applied the framework to a proprietary ALM support environment – Comverse's DiME. These systems are characterized by broad lifecycle support and enterprise-class maturity and stability. They employ the most up-to-date technologies and architecture patterns. In addition, they were designed as integrated ALM environments, unlike many other systems that are a collection of separately designed tools. All these traits make the selected products good subjects to study.

4.1 IBM Rational Jazz

IBM is one of the very few vendors that have solutions for practically every part of the application lifecycle. Some of these solutions were built in-house and some came with acquisitions. The most important of IBM's ALM suites is the Rational product family that contains multiple and diverse tools for development on different platforms, including the well known ClearCase configuration management system (IBM Rational ClearCase V7.1) and a ClearQuest (IBM Rational ClearQuest V7.1)product that facilitates change management and defect tracking.

The most recent IBM's ALM initiative is the development of the Jazz platform and a suite of tools and solutions for application lifecycle management based on this platform. The first such product is Rational Team Concert that provides a collaborative development environment for small to medium sized teams, integrates source control, work item management and build management capabilities. In addition it offers automated real-time reporting and process governance. This section focuses on the Jazz platform and briefly covers the Team Concert suite.

4.1.1 Design goals and architecture

Jazz was designed from the ground up to provide a platform for lifecycle support. Its explicit design goals include:

- Deep task integration across the lifecycle
- Enriched team development support using the latest technologies for collaboration and coordination
- Distributed teams support
- Solutions scalable from small teams to enterprises
- UI integration that meets today's distributed development practice, providing both rich user interface and light Web interface
- Extensibility and openness for other vendors.

In addition, Jazz leverages the Eclipse development platform that has a proven reputation of a stable and extensible platform for development environments. In order to ensure practical usefulness and quality of the system, the Jazz development team used Jazz for Jazz development (self-hosting) (Lemieux, 2008).

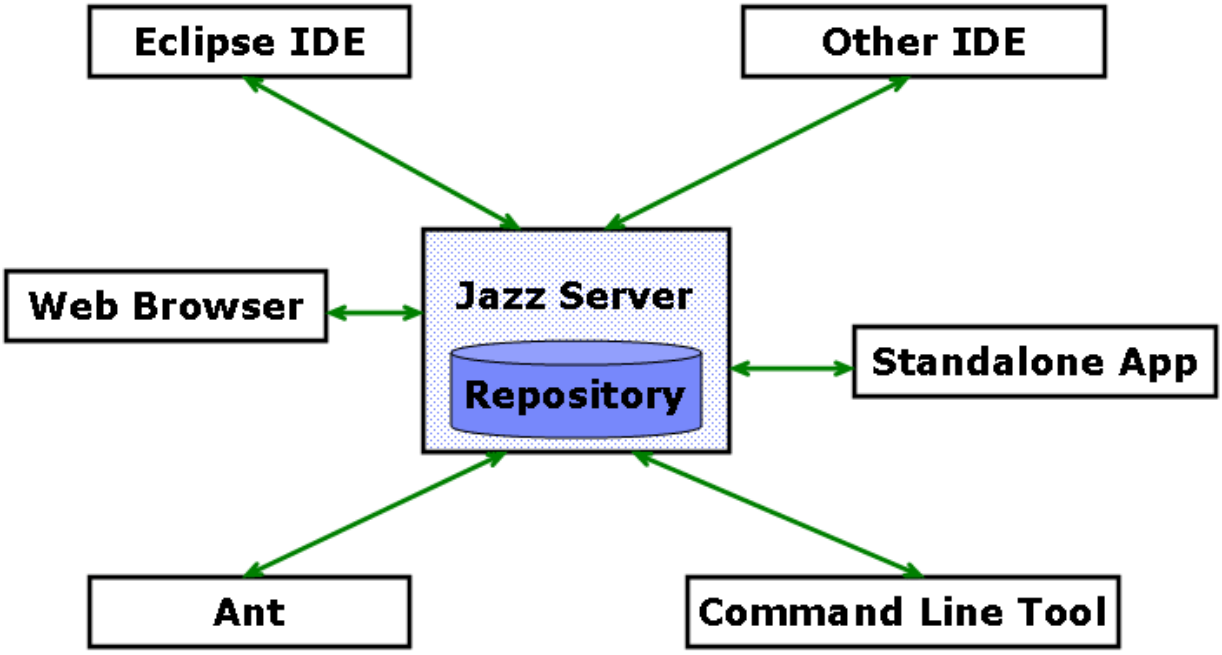

Figure 5. Jazz server and clients (from (IBM, 2008))

Jazz employs the client-server architecture model. A Jazz server hosts a data repository and communicates with clients using Web services over HTTP. The Jazz clients can be of different kinds including integrated IDE plugins, Web browsers, command line tools and Ant[1] scripts. Web access to Jazz client functionality allows working with the server without installing any Jazz-specific software on the client machine.

Jazz employs modular and extensible architecture, which building blocks are called components. A Jazz component supports some aspect of software lifecycle (for example build management). Each component has a server part and a client part that communicate with each other and with their peer components. The components are based on the Eclipse components technology (for Java clients and servers). The components expose functionality that can be invoked by the platform and other components.

[1] Ant – a popular Java-based build tool (Apache)

The Jazz Server is a Java-based web application that currently can run on two application servers: Apache Tomcat for small scale solutions and IBM WebSphere for medium and large enterprises. The Jazz Server relies on the Eclipse's OSGi[2] runtime mechanism for components management and intercommunications.

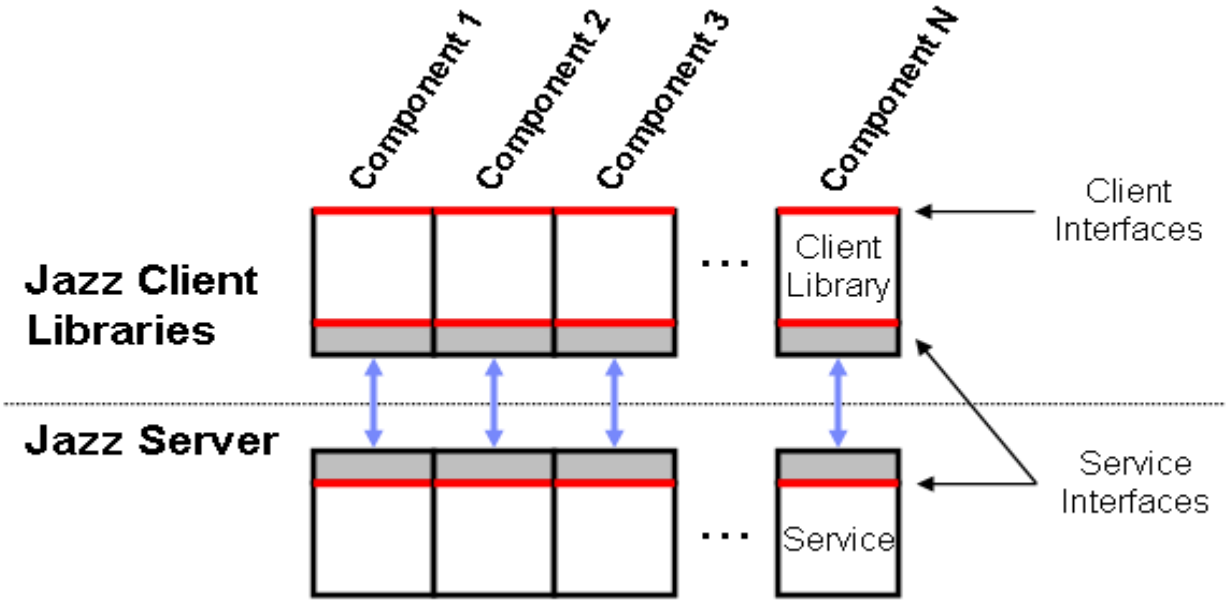

Figure 6.Jazz components (from (IBM, 2008))

The Jazz platform consists of a kernel together with additional components, each responsible for a separate aspect of software development lifecycle. The kernel contains two essential components: Repository and Team Process.

Other components, that are responsible for specific lifecycle activities, build upon the facilities provided by the kernel. The Repository is a central place storing the data of all components. It is based on the data management capabilities of an underlying relational database. The Repository provides to other components a high-level object-oriented access to the stored data, transaction capabilities, auditing of the data items changes and a feed of data change events. The Jazz Repository uses the following extensible and flexible data description mechanism. Components describe their data as a high-level Eclipse Modeling Framework's logical model (Ecore model).

[2] OSGi – an open-source standard specification for a Java-based service platform that can be managed remotely. Eclipse Equinox is one of five certified OSGi implementations. (OSGi Alliance)

This logical model is automatically compiled into a storage model consisting of Java classes that manage efficient storage and transfer of the data items as well as their in-memory representation. The Repository supports a number of enterprise back-end relational databases that provide solutions for repositories of different sizes. The Repository component will be discussed in more details in the data integration subsection.

The second core kernel part, the Process component will be discussed in the process integration subsection.

4.1.2 Jazz Platform Evaluation

In this section we will evaluate the Jazz platform according to our classification framework. Jazz is not an ALM environment, but a platform for such environments, therefore not all parts of our framework will be applicable.

4.1.2.1 Breadth of lifecycle support

Jazz does not directly support any specific part of the software development lifecycle, but provides a platform for tools and components addressing the lifecycle activities. However, in its current state Jazz and its components are mostly oriented towards the code construction part of the lifecycle, such as defect and task management, build management and source control. The existing planning support components are aimed at agile-style lightweight planning of development projects. Another factor that strongly binds Jazz to the development related activities is its Eclipse foundation. The primary user interface to Jazz server and clients is the Eclipse integrated development environment. There is one component that goes beyond the small cycle of development activities – the Jazz Reports. This component will be described later in the chapter devoted to the reporting capabilities.

4.1.2.2 Integration

Jazz was built with an explicit goal of providing high level of integration to its components. Its modular architecture, based on the time-proven model of Eclipse components and extension points, offers a powerful and flexible mechanism for integration. Below we will go over each integration type in Wasserman's classification and see how well Jazz implements them.

Platform integration. In Wasserman's classification, platform integration is "the set of system services that provide network and operating systems transparency" (Wasserman, 1990). Platform integration allows tools to function regardless of their physical distance and the underlying hardware and software configuration. Jazz addresses this by using web services as its communication interface between clients and server. Using this very popular modern technology allows transparent integration of the clients running on different operating systems and separated by thousands of kilometers. Still, the Jazz server and its components are written in Java and this is the only technology acceptable for the server-side Jazz plugins. (It seems that the full potential of the clients not written in Java language depends on the maturity of the Jazz REST[3] Services (Rivieres, 2007) technology, which is under development now (Jonston, 2008). Until Jazz REST services are implemented, the Jazz client-server web services stay internal and can't be accessed directly, therefore there is a need to develop and maintain language-specific client libraries.)

Data integration. The Repository component is a central data hub of Jazz that provides data management and integration services to all other components and their plugins. Each component describes its data in a high level object modeling language of the Eclipse Modeling Framework (EMF). The models are described in XMI (XML Metadata Interchange) and can be created and edited by direct XML manipulation, importing from other modeling tools, such as Rational Rose and by annotating Java interfaces with model properties. The Figure 7 illustrates a definition of a simple class in UML, annotated Java and its XMI representation.

[3] REST – Representational State Transfer. REST is a software architecture approach for distributed systems, is widely used in building Web Services-based API.

```
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    name="library "nsURI="http:///library.ecore" nsPrefix="library">
  <eClassifiers xsi:type="ecore:EClass" name="Book">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="pages"
```

```
/**
   * @model
   */
  public interface Book
  {
    /**
     * @model
     */
    String getTitle();

    /**
```

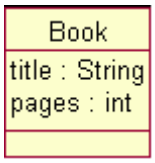

Figure 7. An EMF model definition in UML, XMI and Java (adopted from (Eclipse, 2005))

This high level component's object model is called a **logical model**. Its top-level objects are called **items**. Items can have simple properties (e.g., integers, string) and also **content** properties used to store bulk data (e.g., files). Items can refer to and contains other items. The platform supports auditing and versioning of item's state. Each Jazz Repository item has a universally unique item id (UUID) and can be replicated to another repository keeping its identity.

The EMF engine generates a number of Java interfaces and classes for the defined data model. The generated Java classes contain all necessary information for the framework to learn about the items. This information guides Jazz to build the database tables and the items' internal representation allowing for efficient storage, search and serialization of the items. These generated Java classes form a **component's storage model.** This transformation of the logical model into the storage one is automatically done by the engine.

The Repository provides a generic server-side API for items' creation, fetching, updating and deletion. A component can perform all these operation on the items it created. Another

component's items can be only retrieved. If a component A needs to manipulate component B's items, it can do so only by using the API that the component B exposes.

The Repository provides transactional semantics for data manipulation and implements algorithms for keeping data integrity during concurrent updates. In addition, it supports running complex queries on items based on their properties using the object oriented EJB Query Language (IBM, 2008).

To summarize, the Jazz Repository component provides a generic and powerful mechanism for data management. It possesses a high level of data integration. Because the data items are exposed to the components in a form of Java classes that combine data fields with the behavioral methods we can classify the integration level to be a "semantic" one, when data is associated with the operations that can be performed on it. In addition, the underlying relational database provides a robust and scalable foundation for the Repository. The single repository approach frees Jazz from inter-repository synchronization problems and supplies a natural platform for tools cooperation. At the same time this approach poses problems when it is necessary to connect a tool or a system with its own data storage to a Jazz repository. The Jazz' approach to this problem are **connectors** that should synchronize between the repositories. Currently there are two such connectors coming with Rational Team Concert – one for the ClearCase source control system and another one for the ClearQuest task management system. They will be discussed later.

Control integration. Jazz supports tool collaboration via a paradigm of **events** described below. Components report about interesting changes using a server side API. Examples of such events are build completion, source code item delivery, workspace change, work item status change and many others. The underlying infrastructure backed by the Repository publishes an event feed in a standard Atom or RSS format and this feed can be parsed by any standard reader, including Web based ones. These events are called **ChangeEvents** and carry the following information (Lainhart, 2008):

- *eventItem* – an unique identifier of the Jazz repository item associated with the event

- *eventTitle*
- *eventDescription*
- *eventAuthor* – the user identity of the event creator
- *eventTime*
- *eventCategory*
- and more…

Components can query the change event feed using a standard query API by specifying interesting event parameters.

The Rational Team Concert extends this basic notification mechanism and wraps it into an extensible and generic framework for components' event-based communication. This framework (Pasero, 2008) defines and supports the following terms:

- Event
- Notifier – a mechanism to inform the user about a certain event
- Trigger – it combines a notifier with a family of events under certain condition.

In addition to the events and notifications, the components can invoke each other's public API methods. The invocation can be either Java based or Web service based. This direct components invocation complements Jazz' control integration capabilities with a powerful, scalable and platform independent mechanism.

Presentation integration. The Eclipse integrated environment provides a natural presentation integration platform for Jazz client side components. It supplies templates for such integration and the available Jazz components are built targeting Eclipse as a primary client user interface. Jazz components communicate their data to the user in two main Eclipse views: Team Central and Team Artifact.

An alternative Jazz component user interface is a Web interface that can be accessed via any Web browser and does not require any Jazz-specific software installation on the user machine.

The Jazz Web UI is produced by components' server-side plugins. The Jazz Server puts together the output of different component's and composes a web page that the user interacts with. The resulting web pages are exploiting Ajax and REST web services to communicate with the server-side components' plugins. Jazz provides a development framework for developing both ends of the Web UI.

To summarize, Jazz provides a very good level of presentation integration in both rich and web user interface options.

Process integration. Process support is one of the central ideas in Jazz' design. The Team Process component is an obligatory part of the Jazz kernel, so its services are always available to other components. Jazz' process support is characterized by (IBM, 2008):

- Process-awareness. Process is explicitly designed into the platform and everything happening during the lifecycle happens in the context of some process.
- Process-enablement. Components are governed by the enclosing process and their operations are influenced by the rules and restrictions of this process.
- Persistency. Process and all its attributes are stored in the repository and are treated in the same generic manner as other repository items.
- Flexibility. Process expresses an agreement about how a team's work should be organized. It can be as restrictive or as loose as the team agrees. Even inside the same project a team can customize the process it inherits from its project area (see below).

Jazz represents all aspects of a software project by a **project area**. The project area is stored in the repository and owns all development and maintenance artifacts. The members of the project development teams are also represented in the repository as contributor items that belong to some project area. Each contributor has a role or a number of roles assigned to him in that project. Every project has an associated **team process** that can govern all artifacts, their relationships and activities within this project (IBM, 2008). This governance is realized by making Jazz components *"process-enabled"*. Components do not have any fixed or preferred process type, but rather they allow the governing process to control their actions by:

- Storing component's configuration parameters in the governing process;
- Allowing the process to insert actions before and after the component's operations. For example, a source control component can allow the governing process to verify that the delivered change is associated with some work item and abort the operation, if this is not true;
- Reporting to the process about interesting events. The process can then take an appropriate action upon these changes.

Process is specified in a declarative form (XML) with actions written in Java. Jazz provides templates for a number of popular development methodologies, such as Agile and Scrum. The project team can edit and customize the project process according to its needs.

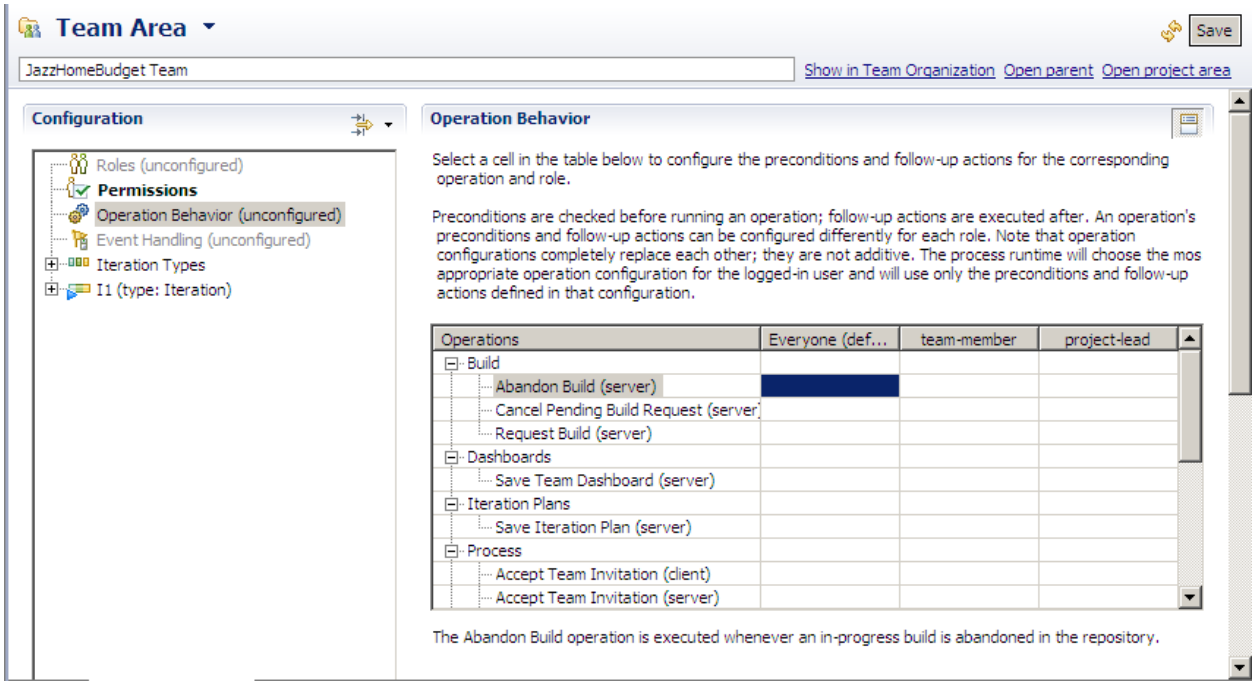

Figure 8. Team Process Customization Dialog

To summarize, Jazz provides comprehensive and deep process support that is seamlessly integrated into every aspect of the project lifecycle. The implemented mechanism is both powerful and flexible and possesses high efficiency and ease of use.

4.1.2.3 Role-based views

Jazz is a platform for building ALM tools and environments, but does not include any such tools. Therefore, supplying a role-based views is not in Jazz's scope. However, Jazz offers a necessary infrastructure for building them. Each team member has a role and his/her permissions to perform different actions are role-based. The roles and permissions are stored in the Repository and the policy is enforced by the Process component.

4.1.2.4 Traceability and reporting

Jazz' single repository architecture facilitates its reporting capabilities. All items and their relationships are stored in a repository, which is built upon a relational database. The Repository component's query functionality allows data retrieval and lookup. The Process component takes care of assuring the linkage between related items, such as source code changes, a work item (e.g., a defect) and a build. A Jazz Team Reports components, which is a part of Rational Team Concert builds upon these capabilities to provide a report engine. This will be discussed in the Team Concert section.

4.1.2.5 Process definition and automation

The Process component discussed in the integration subsection above provides the process automation capabilities.

4.1.2.6 Platforms support

Jazz' natural development environment is Eclipse. Hence its primary programming platform is Java/J2EE. However, Jazz is not limited to this platform only. Jazz can support development in any language and the specific development environment integration should be done by the client side components. While Rational Team Concert (the first available ALME based on Jazz) is built upon Eclipse, a Jazz client for Microsoft's Visual Studio (Lemieux, 2008) is being developed now and will significantly broaden the potential Jazz audience. In addition, IBM is working on integrating Jazz with mainframe environments, such as System I and System Z.

4.1.2.7 Distributed teams support

One of Jazz's design goals is supporting geographically distributed teams' run-time collaboration. The Jazz platform supports distributed development naturally by using web services as its primary communication protocol. The repository data synchronization is provided by the underlying relational database. The collaboration features offered by Jazz allow efficient team members communication regardless of physical distance between them. A practical evidence of Jazz distributed support quality is Jazz's self-hosting: Jazz is being developed and maintained by a geographically distributed team using Jazz.

4.1.2.8 Scalability

This work's goal is assessing an ALME's approach to scalability challenges without going into detailed measurements. The scalability of the Jazz platform can be characterized by:

- An ability to hold and manage **large data sets** common for enterprise development;
- An ability to handle many concurrent users. Server responsiveness should stay within acceptable limits when the number of users grows;
- Network load. The network traffic produced by Jazz client-server communication should not flood the network bandwidth. At the same time, the network communication should provide good response times over WAN.

Jazz addresses the problems related to managing very large data sets by relying on capabilities of the underlying relational database. Jazz supports three database systems: Apache Derby, an open source database suited for small data sets, IBM DB2 and Oracle – for large repositories. The Jazz Repository completely encapsulates the underlying database from upper layers and all storage models and APIs are independent of the chosen database.

Jazz depends upon the hosting application server in managing concurrent requests. Currently it can be installed on two application servers: Apache Tomcat for small environments and IBM WebSphere for large enterprises. Jazz presently supports up to 300 simultaneous users without losing server responsiveness stability.

According to the Jazz' Team Wiki site (Lemieux, 2008), the Jazz developers set an explicit goal of providing good client experience over WAN with slow request round-trip time. The site does not provide network performance benchmarks.

4.1.2.9 Security

Jazz utilizes the hosting application server's authentication mechanism for identity checking. Once identified, the logged-in user is mapped to a representing contributor item and all user's actions can be checked by Jazz components for possessing enough permissions. Jazz can use one of three identity repositories: a simple Apache Tomcat user database, an LDAP server or the application server's internal user management.

4.1.2.10 Incremental implementation

Jazz' modular architecture makes it easy to install only the necessary components. The Jazz server contains only essential kernel modules and an organization can add the required application components. The Jazz server is currently distributed with the Rational Team Concert suite that comes in 3 editions holding different component sets.

4.1.2.11 Integration and interoperability with existing solutions

Jazz can be integrated with other systems using either data import or data synchronization. If an organization has some ALM supporting tool in place, it can either move to a corresponding Jazz component completely, by importing the existing database into the Jazz repository, or use Jazz connectors that will link items in the Jazz repository with the existing tool's artifacts. Such integration solutions come with the products built upon the Jazz platform. For example, a Jazz Source Control component supports integration with Subversion, an open source version control tool. The two integration options are: importing the Subversion repository or linking Jazz work items to Subversion revisions with a help of a Subversion client for Eclipse. More integrations of Rational Team Concert components will be described below.

4.1.2.12 Breadth of vendor support

The Jazz platform is developed by IBM Rational. The first (and currently the only) product suite built upon this platform is Rational Team Concert. A number of IBM business partners are developing products and solutions that integrate with Jazz. They include Black Duck Software, CAST Software, CM-Logic, iRise, Mainsoft, QSM, Ravenflow, SourceIQ, Surgient, VMLogix and WebLayers. IBM is promoting these initiatives by offering to members of its IBM Rational Ensemble program educational resources and services for developing Jazz-based technologies. A number of academic institutions are conducting research projects based on Jazz. At the same time there are no other large vendors working with or planning to support Jazz.

4.1.2.13 Success stories

Jazz became publicly available in the summer of 2008, and the first Team Concert products based on Jazz appeared soon after. So it is too early for industrial Jazz deployment success stories. However, Jazz was developed using Jazz internally and this can be seen as a first and a significant practical success. The developers and management team report (Rich, 2008) about a very positive experience from using Jazz internally and believe that the project's success is due to their first-hand experience as Jazz users. The importance of this self-hosting is not to be ignored, as the Jazz development team represents a globally distributed large team working on an enterprise project.

4.1.3 Rational Team Concert Evaluation

Rational Team Concert is an application lifecycle management suite based on the Jazz technology platform. It extends the integration and collaboration capabilities of Jazz with components supporting different aspects of software development such as planning, configuration management, build management and reporting. Team Concert is the first and currently the only commercial product using the Jazz platform.

4.1.3.1 Breadth of lifecycle support

Team Concert supports and integrates different aspects of the software development lifecycle.

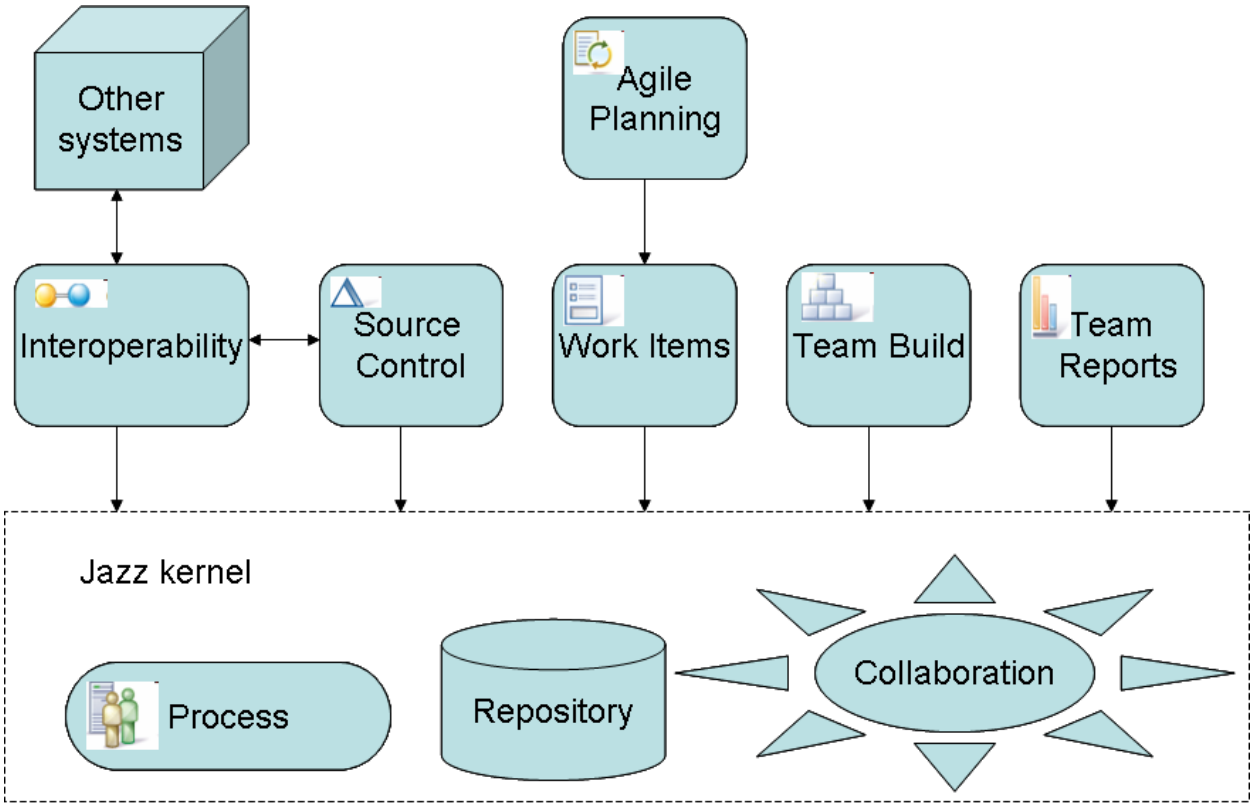

Figure 9. Rational Team Concert Components (adopted from (Rational Team Concert Capabilities))

Its capabilities include (Rational Team Concert Capabilities):

- Work item management. The work item is a basic building block used for task management, work planning, defect tracking and workflow governance. Task work items link together other Team Concert artifacts such as builds and change sets. Work items are stored in the Repository and have a type, attributes and a state. Work items can be tagged, shared with other team members and linked to other work items and artifacts. Work items are customizable and new types of work items can be easily defined. Attachments, such as screen captures and files, can be added to a work item. Team Concert implements an advanced search mechanism that provides duplicate and similar work items search, automatic work item type guessing and other productivity boosting features.
- Source control. The Team Concert's Source Control component is a component-based version control system that has strong support for parallel development. Every source file change is part of a *change set* – an atomic transaction on a set of files. Change sets are linked to work items. Teams and individual contributors share change sets via *streams*.

Parallel development is supported at a number of levels. Developers can work with a local workspace without sharing it with the team while having a full access to all repository features. Work items isolate source code changes in a context of one task. A number of developers can work on a work items isolated from other development. Teams' development is isolated by streams. Streams allow members of the team to share their change sets without affecting other teams. Streams support hierarchy allowing code sharing between teams when integration is required. The Source Control component is fully integrated with defect tracking, builds and process automation.

- Agile planning. The Agile Planning component provides tools for development iteration planning. It allows selecting tasks for the iteration and monitoring their progress and developers' work load. The Agile Planning component is highly integrated with the Work Items and the Process components. However, it is not suitable for planning and tracking large development project, but is rather oriented to iterative, agile-style development.
- Builds management. The Team Build component integrates builds with other development artifacts, such as process, work items and change sets. The component works with existing command line build scripts and tools (Ant, Maven and others). Builds are linked with the change sets and work items they include. Builds can be requested and scheduled, compared and tracked. The Build component provides automatic reporting on builds characteristics such as failure rate and duration.
- Reporting. The Team Reports and Web Dashboards components provide visibility into project health. They offer both real-time status and historical perspective on tasks, defects, builds and source streams. This information is presented both in the rich client Eclipse and web user interface. The components supply a large number of predefined report templates suitable for tracking the status of projects of different types. The reports and report templates can also be customized. Historical data is stored in a special database, *data warehouse*, which is designed to store, query and discover trends in aggregated data. The data warehouse is populated automatically from the data stored in the repository. Dashboards provide a single place representing project overall status and its different aspects (iterations progress, work items statistics, event log), in an easy to understand and use web interface.

- Interoperability. Two Connector components allow Team Concert to interoperate with other Rational products. The ClearCase Connector preserves synchronization between a Team Concert Source Control stream and a ClearCase stream. This allows Team Concert users to work on a ClearCase stream. The operation is not transparent as it requires explicit synchronization request to be issued, but it can be automatically scheduled thus reducing user's involvement. According to the latest news from IBM, it plans to move ClearCase onto the Jazz platform. The ClearQuest Connector synchronizes between ClearQuest records and Team Concert work items allowing teams to use both tools simultaneously.
- Requirements management. Requirements Composer (in the beta stage of development currently) is a new Team Concert member that brings in requirements definition and management capabilities. It brings the functionality of the popular RequisitePro tool into the integrated Jazz environment. Its abilities include multiple requirements sources consolidation, use case development, user interface prototyping and many others. In addition to that it is tightly integrated with the process and work items.
- Test management. Quality Manager (in the beta stage of development now) adds test planning, execution and tracking capabilities. The tests are tightly integrated to requirements and work items, such as defects.

To summarize, Rational Team Concert provides a rich and integrated set of tools for all major development activities and is gradually extending into the adjacent areas of the lifecycle – requirements and test management. It also can interoperate with a number of other popular lifecycle management tools.

4.1.3.2 Integration

Team Concert inherits its integration capabilities from Jazz. All components work in a tightly integrated manner sharing the data through the shared repository. This allows developers to perform all lifecycle activities in a single integrated environment and all these activities are interrelated and governed by a team-predefined process.

One important Team Concert's addition to the Jazz integration characteristics is the Open Services for Lifecycle Collaboration initiative (Open Services for Lifecycle Collaboration). This initiative's goal is to allow collaboration of tools from different vendors without creating tool-to-tool integrations. The initiative sees an ideal solution to this problem as a *"uniform architecture and set of protocols that allow resources from loosely coupled tools to be integrated in a consistent way"*. The resources are the lifecycle artifacts, such as defects, test definitions, source code and others. The initiative proposes three fundamental principles of such solution:

- A universally unique and globally accessible resource address. Specifically, the Open Services propose URL as such addressing mechanism.
- Semi-transparent XML-based resource format. Tools agree about common elements of the resource format and can change their private elements format and meaning without breaking other components.
- A common resource access protocol. The Open Services adopt RESTful web services for this purpose. Discussion of these services is out of the scope of this work.

In addition to these principles, the Open Services initiative proposes a typical set of lifecycle resources and their relationships. The agreement on these relationships is another essential component of the independent tools collaboration. Many of the Team Concert components already employ or are migrating to the Open Services architecture. For example, Rational Requirements Composer interoperability is fully based on Jazz REST Services.

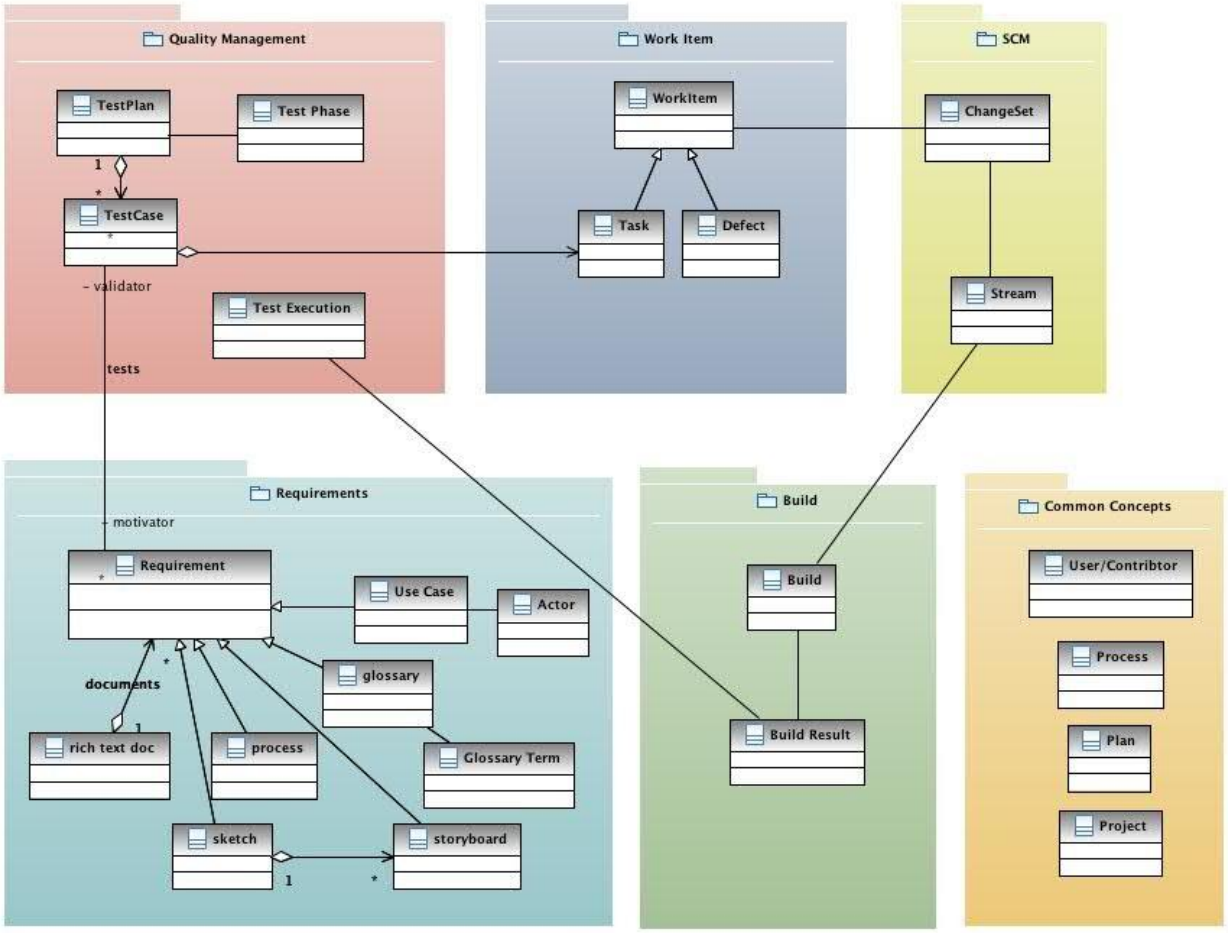

Figure 10. Open Services Resources and Relationships (from (Typical Lifecycle Resources and Relationships))

4.1.3.3 Role-based views

Team Concert inherits from Jazz a flexible built-in mechanism of roles and their permissions. All users are assigned one or more roles in the project. Each role has a set of permissions to carry out certain actions. This scheme controls user actions inside the project but does not restrict user's access to different areas of project data. The author could not find a way to define, for example, that a user that has only a "Builder" role is not allowed to access the project overall progress view. Maybe, this is done on purpose – Jazz's designers believe that sharing as much information as possible is a positive thing. On the other hand, the Team Concert client interface is highly customizable and a user can select the views he/she is interested in. For example, the

Team Central Eclipse view is built from sections that include My Open Work Items, Event Log and Team Load. Sections can be added, removed and further customized. Thus Team Concert provides flexible and adjustable views, but does not enforce restrictions on the information access. As for the view for the non-development part of organization, the Dashboards component provides high-level and aggregated information about project.

4.1.3.4 Other technical aspects

The abovementioned Team Reports component provides powerful and flexible reporting capabilities. The Reports component uses data warehouse for historical data storage and processing. Its report engine is based on the Eclipse Business Intelligence and Reporting Tools. The reports are created from templates and their data set is defined by the user-supplied parameters. The reports can be presented in various user interfaces: rich client, web or integrated into the Dashboards view.

Team Concert can be extended via two mechanisms – Jazz components and REST services. New functionality can be tightly integrated into the Jazz platform by implementing a server plugin that will run inside the Jazz Server and a client plugin that will provide the front-end user interface. Such components are being developed by a number of IBM business partners that participate in the Rational Ensemble program. The Team Concert Document Collaboration developed by Mainsoft is the first Team Concert component created outside of IBM. The Open Services approach suggests a platform for a resource-based loosely coupled tools collaboration. In the long term this approach can allow an organization to have tools from different vendors working together without being heavily dependent on each other. This direction looks very promising; however its success depends on the number of other large vendors that will participate in the initiative. So far the author could not find any evidences of other major players in the ALM domain that support the Open Services initiative.

The process automation, platforms support, distributed teams support, scalability and security aspects of Team Concert are the same as these of Jazz and were discussed above.

4.1.3.5 Incremental implementation

Rational Team Concert is available in three different editions, but they mostly contain the same functional components, and differ in the number of supported users. So an organization interested in only some of the Team Concert components is still required to pay for the entire packet.

4.1.3.6 Integration and interoperability with existing solutions

Team Concert presents a number of interoperability solutions for popular tools and systems. As mentioned above, Team Concert (in its most expensive Standard edition) provides connectors for ClearCase and ClearQuest. It can also link work items to revisions of a popular open-source Subversion version control tool (in all editions). More integrations are being developed: already mentioned Team Concert Client for Visual Studio and the Mainsoft's SharePoint Document Collaboration Component.

4.1.3.7 Breadth of vendor support

A number of IBM business partners listed above develop Jazz and Team Concert based systems and extend them with additional components. As the same time, no other large independent vendor has joined these small companies.

4.2 Microsoft's VSTS

Visual Studio Team System (VSTS) is an integrated ALM solution from Microsoft. It comprises tools suites that work collaboratively sharing a common data repository and guided by a common process. VSTS provides impressively wide support for many life cycle aspects and is especially strong in the code construction activities. The VSTS platform includes integration with the Microsoft Office product suite, thus exposing its functionality to non-developer users, such as project managers and business analysts. The product was released more than three years ago, it has a wide user audience and a lot of VSTS implementation case studies are available. In this section we will describe the VSTS' relevant design and architecture details and will evaluate it using our classification framework. VSTS and Jazz share a lot of common concepts in

terminology, architectural patterns, supported functionality and extensibility techniques. Both platforms employ client-server architecture and use web services for inter-component communication. Both are mainly targeted at development activities and provide very similar sets of features. Therefore this work will cover these common characteristics briefly as they were described in more detail in the section devoted to Jazz.

4.2.1 Architecture

VSTS consists of a server and a suite of client product editions (Microsoft, 2009).

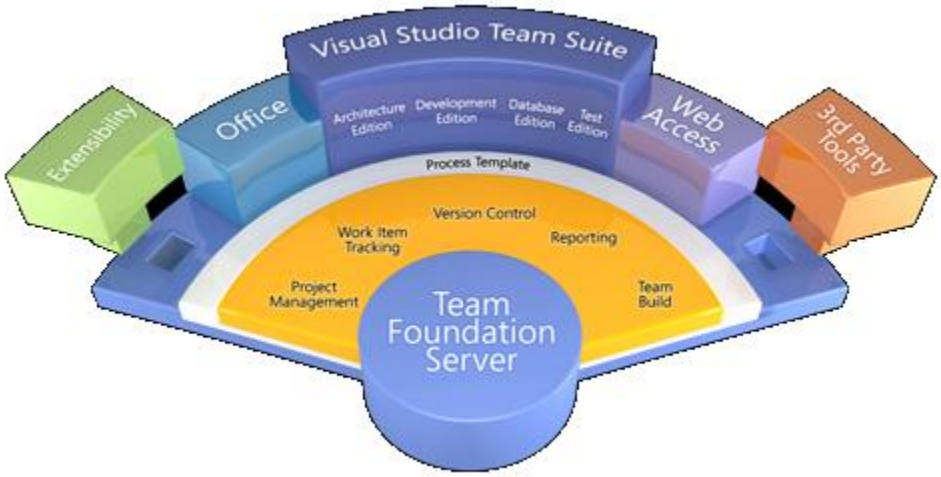

Figure 11. VSTS architecture (from (Microsoft, 2009))

Team Foundation Server (TFS) has two main functions: being a central hub of all VSTS data and hosting all VSTS functional capabilities. Logically it consists of two tiers: a *data tier* that stores all persistent data of all Team System tools and an *application tier* that hosts all TFS functionality (version control, work item management, project management and others). Client applications communicate with the application tier via web services. The application tier accesses the data tier using database connections. The two tiers can reside either on the same physical server or separately. The data tier consists of operational stores of all Team System tools and a hybrid (relational and OLAP) data warehouse that is used for reporting. The operational stores and the warehouse are MS SQL server databases.

VSTS exposes its functionality to the user in a number of ways. First, most of this functionality is available in the Visual Studio IDE. For project managers and other non technical project members accessing the project data via the development environment is confusing and overloaded with irrelevant technical details. A special simplified UI called Team Foundation Client exposes access to such features as project progress monitoring and work items tracking. In addition, VSTS exposes a Web access to its data via the SharePoint project portal. MS Office applications can also access this data using the SharePoint integration. Finally, all application-tier web services are public and custom client applications that access them directly can be created.

The VSTS architecture allows outside tools to integrate into all levels of the system.

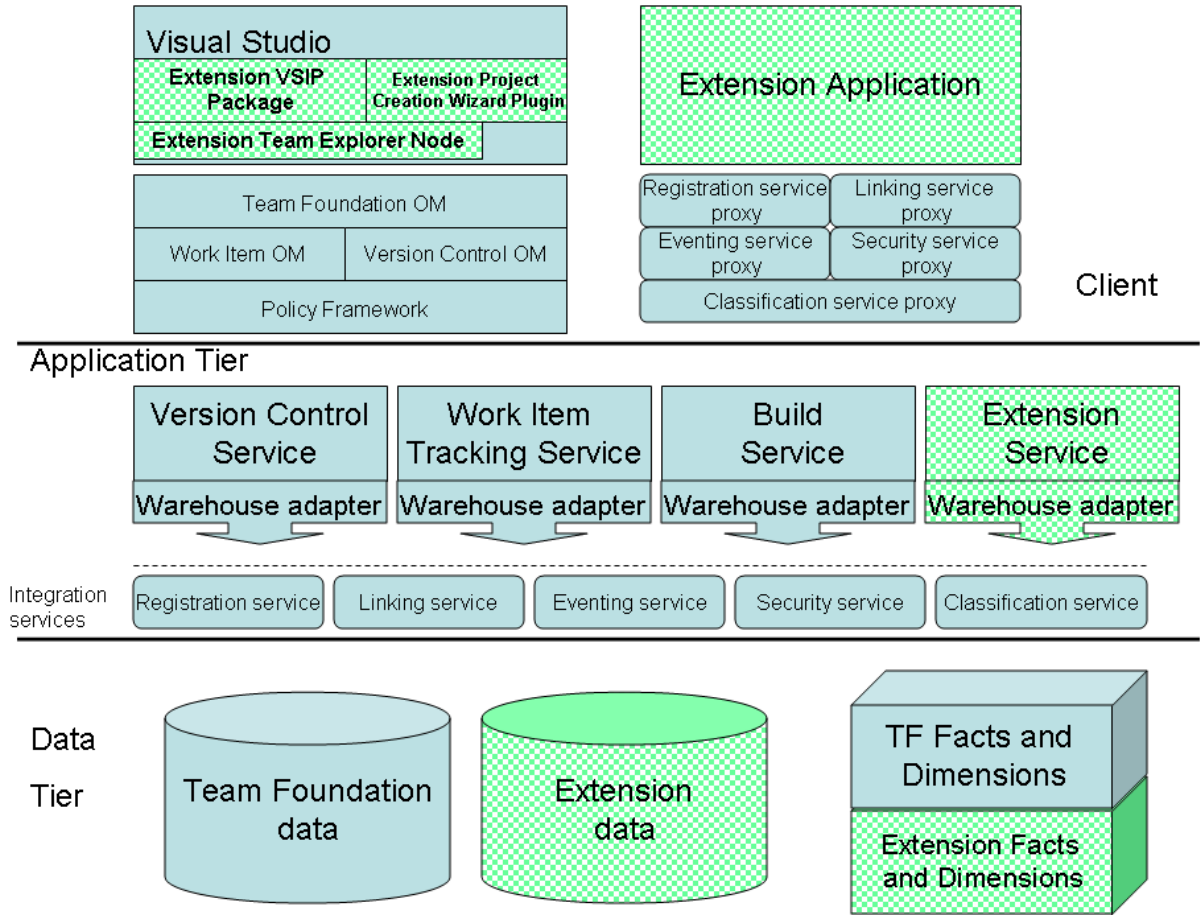

Figure 12. VSTS Architecture with an Integrated Extension (adopted from (Minium, 2006))

The Figure 12 represents a custom extension application integrated into VSTS. VSTS exposes a set of integration services (a registration service, a linking service, an eventing service, a security

service and a classification service) that allow full integration into the application tier. These services will be covered in more detail later.

4.2.2 VSTS Evaluation

4.2.2.1 Breadth of lifecycle support

VSTS has evolved from the Visual Studio integrated development environment and hence provides very strong and diverse support for development activities. The functionality that VSTS offers to developers goes beyond the support provided by most other systems, both integrated and standalone. The Team System Suite includes a number of client product editions each one specializing on different aspects of the development and testing activities.

- **VSTS Architecture Edition** helps architects, operational managers and developers in distributed systems design.
- **VSTS Database Edition** provides database administrators and developers with instruments for database change management and testing.
- **VSTS Development Edition** contains powerful tools for developers. These include static code analyzers, code coverage and performance profilers, code refactoring tools, tools for insecure code identification and automated unit testing.
- **VSTS Test Edition** is a suite of tools for Web applications and Web services testing. These tools allow testers to define, execute and manage tests. The support includes load-test management.

All VSTS editions work in tight integration with the functionality provided by the application tier. This functionality is the core of team collaboration, tying together all major activities around the actual coding. These capabilities include:

- **Version control** – TFS exposes an industrial configuration management functionality that is backed by MSSQL Server and provides source code version control and document management.

- **Work items tracking –** TFS provides functionality for managing work items, such as defects, tasks and requirements. The work items are stored in XML format and can be extended with new fields and attributes. New work item types can be easily added to the system. Work items can be accessed and modified both from the Visual Studio IDE and from MS Office applications (Excel and MS Project).
- **Build management** – TFS enriches the MSBuild engine with Team System integration capabilities, such as automated testing and work items update. Builds results are stored in the data tier and can be queried and analyzed.
- **Reporting** – TFS reporting capabilities build upon SQL Server Analysis Services and SQL Server Reporting Services. The VSTS tools' data is transformed by warehouse adapters and is stored in the data warehouse SQL Server. The reporting engine provides aggregated and cross-product reports, such as bug trends, test coverage and code churn. Custom reports can be generated using third-party tools and MS Office applications.
- **Project management** – TFS binds together all data, tools and processes used in the development lifecycle of a software application using a concept of a *team project* (Microsoft, 2009). The team project isolates all artifacts relevant to the software application under development. These artifacts include: work items (bugs, assignments and requirements), code, test definitions and test execution results, metrics and documentation. The team project may also include different policies (e.g., source control permissions), team project reporting site and the team project portal (based on SharePoint Services). The project's process is governed by a *project process template*. The process template contains definitions for project work items types, roles and permissions, document templates and predefined reports. TFS supplies two predefined process templates: one for Agile-style development and the second for a more formal development process. More process templates are available from the third parties. An organization can create its customized process template. The process support will be discussed in more details later, in the integration subsection. The team project allows cross-tool reporting on a single project, as it naturally isolates the project's artifacts. The team project definitions and data can be accessed via a Team Explorer view window of

Visual Studio and via Team System Web Access. The project SharePoint portal provides access to the project documents, document templates and health reports.

- **Extensibility –** TFS builds upon a set of tool integration services. These services are called *Shared Services* and are used by outside tools to extend TFS and fully integrate into the TFS environment. The Shared Services will be described in more details later.

4.2.2.2 Integration

Platform integration. VSTS provides platform integration by means of its Web services API to the application tier. These Web services allow custom tools and products to use the functionality of Team Foundation Server regardless of the underlying hardware and software environment. A number of commercial products exploit this mechanism. Teamprise Client Suite (Teamprize, 2008) allows developers to use all TFS features from Eclipse and from a number of other IDEs (JBoss, Adobe Flex Builder, Rational Application Developer). It allows accessing the TFS services from non-Windows operating systems, including Linux and Mac OS X. The Teamprise product itself is written in Java, which emphasizes the excellent platform integration capabilities of VSTS.

Data integration. The *Linking Service*, which is a part of Team Foundation Core Services, provides the data integration facilities to VSTS components. It allows separately designed tools to link their artifacts without having a common database. For example, in VSTS 2005 the work items tracking tool and the version control tool use this service to link work items to source control artifacts.

VSTS' basic data entity is the *artifact*. Tools are *artifact providers*. An artifact provider registers the artifacts types it offers and must implement a set of interfaces that allow other tools to read those artifacts and refer to them. Each artifact has a unique URI that is used by the linking service to address it. Figure 13 represents the VSTS artifact URI format.

```
vstfs:///<tooltype>/<artifacttype>/<tool-specific id>
```

- **Vstfs** A constant that may be used as a custom protocol. In version 1.0, it is converted to a URL and made widely available that way.
- **<tooltype>** Identifies the tool interface supported by the tool responsible for maintaining and answering questions about the artifact—in other words, the artifact provider. This enables a calling tool (one holding a link) to decide how to handle an artifact based on the interface offered by the tool. For example: vsbug, vsversionstore, and so on. Tooltype identifies the API and is specified by the tool.
- **<artifactttype>** The type of artifact that the tool maintains. A tool only supplies an artifact type if the type is immutable. Artifact types are registered by the tool at installation time.
- **<tool-specific id>** An immutable reference to the artifact instance. The tool creates the tool-specific ID and maintains it.

Examples

- A URI pointing to a file stored in source control.
  vstfs:///vsversionstore/file/b4e3216
- A URI pointing to requirement record in work item tracking.
  vstfs:///vstfArtifact/vsworkitems/req/345

Figure 13. VSTS URI Format (from (Microsoft, 2009))

The artifact provider also registers the *ILinkingProvider* Web service interface that implements a **GetArtifacts** method. This method is used to retrieve artifacts of the types this provider owns. The provider must raise events when its artifacts change.

Artifacts are connected to each other by *links*. During the link type registration the link hosting tool tells the system the referring and referred artifact types and the inverse link type. The tools that refer to other tools' artifacts are known as *artifact consumers*. An artifact consumer must implement the *ILinkingConsumer* interface which has a single **GetReferencingArtifacts** method. The consumer must also respond to **ArtifactChanged** events updating the links it hosts. This very generic mechanism allows artifact providers and consumers to maintain the linkage between their artifacts without knowing anything about each other. Thus VSTS employs the **repository-neutral** paradigm, where each tool is responsible for managing its data, and the data synchronization relies on the tools' conformance to the defined interfaces and conventions. This system behavior is called *loose coupling*. VSTS also supports *tight coupling* when the artifact consumer links to the artifacts using some knowledge about specific APIs and implementation details of the artifact provider. This linking gives more control, but comes at the expense of strong dependency between the components. To summarize, VSTS' data integration mechanism is both flexible and powerful, and allows tools to integrate with each other without creating

brittle interdependencies. Its integration level is clearly semantic, as Web service interfaces associate functionality with data types.

Control integration. The Team System's *Eventing Service* serves as the main control integration mechanism. VSTS implements a reliable publish-subscribe infrastructure that allows tools to register event types they publish. Tools, services and users subscribe to the interesting events and get notified either by e-mail (users) or by web services notifications. Event types are described in XML Schema Definition. When subscribing to events notification, the event consumer can specify a filter expression. VSTS supports a powerful filtering specification that uses the Visual Studio Event Filtering Language modeled after SQL expressions. In addition to events, tools and services can directly access other tools' object model (the abovementioned tight coupling). Events publishing and subscription can be done using Web services, thus providing platform independence.

Presentation integration. The main VSTS components' user interface is the Visual Studio integrated development environment (IDE). The Team System tools use the IDE's integration platform in order to get access to all visual components of the environment and to integrate into control and navigation elements of the IDE, such as menus, toolbars and views.

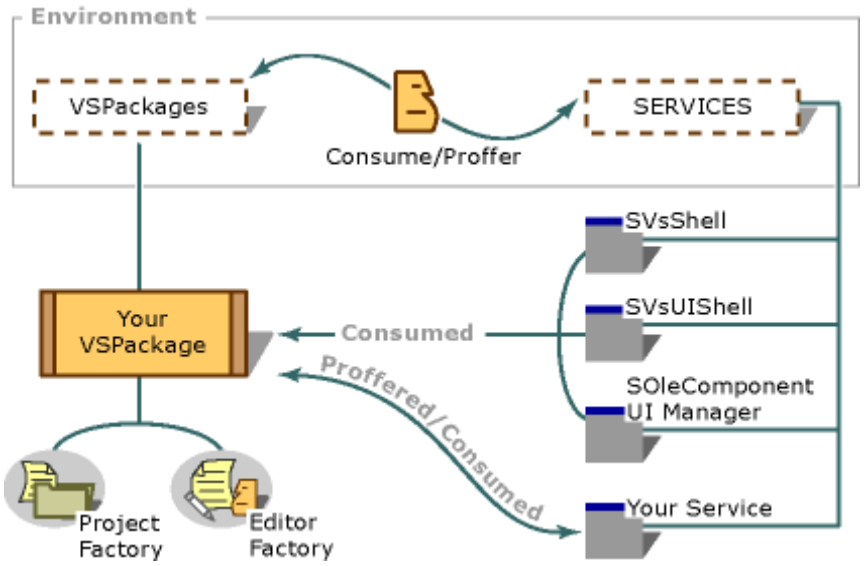

Figure 14. Visual Studio Extensibility (from (Microsoft, 2009))

The Visual Studio IDE hosts software components in form of *VSPackages.* VSPackages can access all IDE's UI via programmable COM and .NET interfaces. The packages expose services to the environment and other packages and use services provided by them. A Team Explorer view is a focal presentation point of all Team System's projects and their data: work items, builds, documents and other artifacts. The Team Explorer can be extended by writing custom plug-ins that provide additional functionality.

Process integration. The *process template* is the main instrument of VSTS process support. The process template defines process-related aspects for each individual tool and supplies process guidance materials. The process template is used by VSTS in conjunction with tool-specific *Project Creation Wizard plug-ins.* These plug-ins (whose names and interdependencies are specified in the process template) are invoked during project creation. Each plug-in sets up its data types and other artifacts according to the template. For example, the Version Control plug-in sets up permissions for operations such as Read, Checkin and Label for different user roles (Contributor, Administrator and Builder). These roles and their project-level permissions are also part of the process template. The XML snippet below represents a part from the permissions settings of the Agile process template. It contains the permissions given to the members of the Contributors group on different types of entities (e.g., project).

```
<group name="Contributors" description="Members of this group can add, modify, and delete items within the team project.">
<permissions>
<permission name="GENERIC_READ" class="PROJECT" allow="true" />
<permission name="PUBLISH_TEST_RESULTS" class="PROJECT" allow="true" />
<permission name="GENERIC_READ" class="CSS_NODE" allow="true" />
<permission name="WORK_ITEM_READ" class="CSS_NODE" allow="true" />
<permission name="WORK_ITEM_WRITE" class="CSS_NODE" allow="true" />
<permission name="START_BUILD" class="PROJECT" allow="true" />
</permissions>
</group>
```

Figure 15. Permissions settings of the Agile process template

The template also contains a Sharepoint section. The template's Sharepoint section contains a lot of descriptive material that covers in depth all aspects of the process. It also contains numerous document templates, sample project management materials (MS Project's project plan, a project checklist, test development plan), and other process guidance documents. The project template contains many useful reports and work items queries.

One of the central pieces of the process template is the working items description. It contains definitions of the typical work item types used in the described process. The work item description contains:

- **Fields**, their names, types, description and value ranges
- **States and state transitions.** State transitions include:
    - The original and target states
    - The transition's reason
    - Changes to the fields that result from the transition
    - Actions that accompany the transition.

The actions are described as .NET managed methods that should be executed when the transition takes place.

VSTS' process integration support is mostly descriptive and static. It sets up the scene for the process implementation and supplies the team with all necessary templates and explanations. However the template's influence on the run-time system behavior is quite limited. The only place the process is automatically enforced is the work items actions. It seems that compared to Jazz, the process awareness of the VSTS components is low.

4.2.2.3 Role-based views

VSTS support for role-based views is both rich and powerful. This support has two aspects: permissions and views. The permissions model of VSTS is based on a discretionary access control (DAC) to the parts of data and functionality for individual users, user roles and groups. The project's process template defines groups of users for the project and sets the high-level permissions for each group. For example, it can grant members of the Readers group the

WORK_ITEM_READ access, and provide the member of the Contributors group with the WORK_ITEM_WRITE and START_BUILD permissions. The VSTS tools can further specify permissions on specific actions and operations. This is also a part of the project template. Project administrators can set users permissions on a project basis or on a server basis.

VSTS provides a number of product editions targeted at specific VSTS user audiences. The Architecture Edition supplies architects and operations managers with a suite of tools for design and validation of service-oriented solutions. The Database Edition contains advanced tools for database professionals. The Test Edition provides a suite of testing tools. All these editions are fully integrated with the common core functionality of TFS, such as work items and version control. These specialized editions provide the targeted practitioners with the relevant tools while keeping them in tight collaboration with the rest of the team.

In addition, VSTS provides plug-ins to Microsoft Project and Microsoft Excel that allow project managers to access TFS's work items management capabilities through these tools, thus hiding the complexity and the irrelevant features of the IDE. Finally, VSTS 2008 provides a standalone rich client application, Team Explorer (called Team Foundation Client in earlier versions), that exposes the TFS services outside of the Visual Studio IDE. Team Explorer enables users of other development environments to enter the integrated lifecycle supported by the TFS.

4.2.2.4 Traceability and reporting

Team Foundation Server provides the reporting capabilities based upon SQL Analysis Services. Each TFS component (version control, work items tracking, etc.) keeps its data in a separate relational database. In order to ensure efficient data querying VSTS maintains a specially designed data warehouse that combines a relational database and an Online Analytical Processing (OLAP) cube. The components' data is pulled from the tool's database, aggregated and then placed into the warehouse. The reports are generated from templates. Each process template contains a number of predefined report templates. New reports can be created by modifying the existing ones, using filters on existing reports or with Microsoft Excel and Report Server.

The next version of the VSTS will contain requirements management functionality. This new feature will include requirements traceability.

4.2.2.5 Process definition and automation

VSTS' process automation support was described earlier in the Integration subsection. In short, it seems that the process automation is not integrated well enough into the VSTS design. The project process template sets up an environment for the process, but these settings have a mainly static and descriptive nature. The runtime behavior of only some components is governed by the process, and there is no single and standard way to make components process-aware and process-enabled.

On the other hand, the project templates mechanism together with the VSTS extensibility features can provide a platform for further lifecycle automation. One such solution is APPRISE (Adaptive Project and Process Regulation and Information System for Enterprise) developed and successfully deployed by Hewlett-Packard (Microsoft, 2005).

4.2.2.6 Platforms support

VSTS' primary server and client operating environment is Microsoft Windows. VSTS is the most popular IDE for Windows applications in different programming languages and frameworks (.NET, ASP and others). At the same time the Web Services interface to the VSTS application tier makes it possible to use VSTS for development in other development and runtime environments, and even other client operating systems. A large number of commercial products leverage this technology to bring the functionality of VSTS to UNIX, Linux, Mac OS, Mainframe operating systems as well as Java/J2EE and embedded systems application development (Microsoft, 2008).

4.2.2.7 Extensibility and openness

As shown above, VSTS can be extended in each of its three tiers: data, application and client. Extending VSTS in the client tier is achieved by using the Visual Studio Integration Package

technology. The client tier extension can access the object model of all components of the VSTS application tier and becomes a part of the integrated environment. The application tier can be extended using the integration Web services. This approach allows integrating into VSTS components written in other languages and running on other operating systems.

The cornerstone of the VSTS extensibility is the *Registration Service* (Meier, Taylor, Bansode, Mackman, & Jones, 2007). It provides a central database of all services' metadata. Services can discover each other using this database. The Registration Service supports loose coupling for both client and server components. A tool enlists itself into the registration database using an XML file that describes the tool's attributes, the services the tool exposes and the artifacts it hosts. Using a simple XML format and a set of standard conventions tools can communicate without being bound to any specific implementation language, runtime platform or even HTTP protocol implementation. Figure 16 represents an extract from a sample tool's registration XML that describes the tool, its linking provider service and its artifacts.

```
- <RegistrationEntries>
  - <RegistrationEntry>
      <Type>Collectibles </Type>
      <ChangeType>Change </ChangeType>
    - <ServiceInterfaces>
      - <ServiceInterface>
          <Name>LinkingProviderService </Name>
          <Url>http://[AT]:[PORT]/CollectorService.asmx </Url>
        </ServiceInterface>
      + <ServiceInterface>
      + <ServiceInterface>
      </ServiceInterfaces>
    + <Databases>
      <EventTypes />
    - <ArtifactTypes>
      - <ArtifactType>
          <Name>Collector </Name>
        - <OutboundLinkTypes>
          - <OutboundLinkType>
              <Name>Holds </Name>
              <TargetArtifactTypeTool>Collectibles </TargetArtifactTypeTool>
              <TargetArtifactTypeName>Artifact</TargetArtifactTypeName>
            </OutboundLinkType>
```

Figure 16. Sample tool registration data

While exposing these powerful extensibility features, VSTS does not open its source code or internal interfaces. The Web services of the application tier components (e.g., Version Control Web service and Work Item Tracking Web service) are not documented and are not encouraged to program against. Instead, the third-party integrators are provided with a rich managed object model that exposes access to these services in the client tier.

4.2.2.8 Distributed teams support

The Web services interface to the VSTS application tier makes the location of TFS transparent to the clients. Thus VSTS naturally supports distributed development. In addition, TFS includes a special architectural component, called Team Foundation Server Proxy, which manages downloads and file caching at the remote site. The TFS proxy significantly reduces bandwidth usage and improves the performance of remote operations, while staying transparent to the clients.

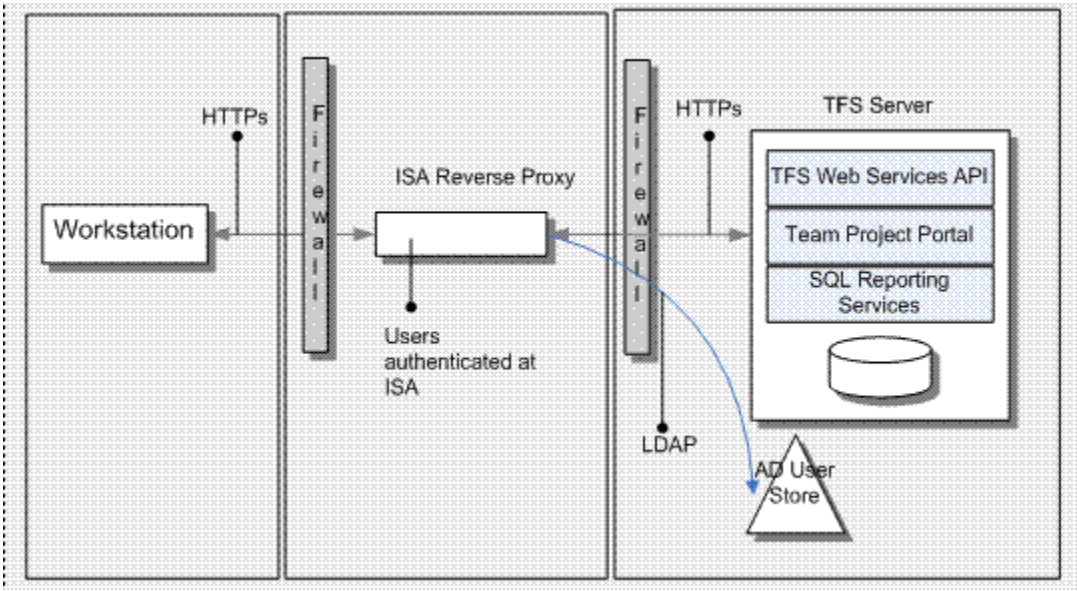

Figure 17. Accessing TFS through a reverse proxy (from (Meier, Taylor, Bansode, Mackman, & Jones, 2007))

It is possible to configure TFS to use HTTPS and VPN connections for remote users. VSTS also allows Internet-based users to authenticate using HTTP basic authentication and connect to TFS even without VPN. Still, there are important questions in the distributed development support that VSTS has not solved yet. For instance, there is no good solution for synchronizing two TFS repositories.

4.2.2.9 Scalability

TFS three-tier architecture allows it to scale very well. The data-tier backed by MS SQL Server repositories can hold very large data sets. The Web server's application-tier manages multiple user requests. VSTS is a quite mature product and a number of large enterprise VSTS

installations exist, including the Microsoft-internal TFS that serves the VSTS development (this process is called “dogfooding” at Microsoft). These installations demonstrate VSTS scalability.

The TFS development team periodically conducts scalability tests of the product and publishes the hardware recommendations for different team sizes. The tests’ ultimate goal is to find a configuration that can serve requests without losing responsiveness. The definition of “responsiveness” in these tests is a little vague, as it does not specify a concrete round-trip time. Instead, the team looks at the graph representing the dependency of the response time from the server load. The point of the graph where the response time starts growing faster then the load is called “the knee”. The tests’ goal is finding the configuration that will serve the specified team without reaching the “knee” and not going above 80-85% CPU utilization. The server load is modeled after the load patterns of the internal “dogfood” DevDif TFS server that has been actively used and monitored for a number of years. Below are the latest hardware recommendations for the TFS hardware configurations. They illustrate the scale abilities of TFS.

The data set of the largest load tested configuration included 10,000,000 files, 250,000 work items, and its average workspace size was about 20,000 items.

Probably the best practical example of TFS scalability is the internal “dogfood” implementation. According to the latest data presented at the TechEd 2008 (Saad, 2008), there are over fourteen thousands active users of TFS in Microsoft, working on more than forty million files and more than three million work items. The total data size of the TFS servers inside Microsoft is over seven terabytes.

4.2.2.10 Security

VSTS and TFS have a strong security model in place. This security model has three components: topology, authentication and authorization. **Topology** deals with servers’ deployment and network protocols between the tiers. **Authentication** verifies the credentials of a user or a process. VSTS employs Windows integrated authentication. **Authorization** checks the user’s permissions to carry out the requested action. TFS authorization mechanism makes use of TFS

users and groups as well as Active Directory users and groups. The security permissions can be controlled on a per-project basis or a server-wide basis and can be further refined by individual tools (version control permissions and work items permissions).

4.2.2.11 Incremental implementation

VSTS comes in a number of editions, but they contain mostly the same TFS features and differ in the specific development activities toolset. This forces the organization to purchase all components of the TFS regardless of its actual needs. A number of VSTS users complain about this lack of the possibility to adopt VSTS incrementally (the author has found a number of such questions and requests on the Internet). The "Team Development with Visual Studio Team Foundation Server" book also called "TFS Guide" (Meier, Taylor, Mackman, Bansode, & Jones, 2008), which contains patterns and practices of VSTS adoption, was published recently. However, the author could not find there clear instructions for gradual adoption process as well.

At the same time, the process of gradual transition to TFS has happened inside Microsoft itself. Different Microsoft's divisions have adopted VSTS internally (Saad, 2008). The Developer division has pioneered this adoption in 2005 with a limited set of functionality and by 2008 the entire division was using all TFS features. The Windows division started in 2007 to use the planning features and is going to migrate to TFS-based bug tracking. During this transition the development organization is actively using the migration tools and mirroring solutions that synchronize VSTS tools with their legacy counterpart systems. These mirroring and synchronization tools (available in VS2008 and VS2010) will be described in the next section.

To summarize, while there is no option for an organization to purchase only the parts of VSTS it is interested in, a phased or partial migration to VSTS is possible. However, there is some lack of materials describing this transition process.

4.2.2.12 Integration and interoperability with existing solutions

VSTS' Migration Toolkit (TFS Migration and Synchronization Toolkit, 2008) provides means for interoperability with other ALM systems and for migration from other systems to VSTS. This

toolkit is available as a shared open source. It allows both replacement of other version control and work items tracking systems with TFS and bidirectional synchronization between TFS and these systems. Other solutions include Microsoft's tools for migration from Rational ClearCase, ClearQuest and Microsoft Visual SourceSafe. Finally, a number of third-party companies, such as Notion Solutions, Accentient and Persistent Systems offer services for data import from existing defect tracking, help desk and project management systems into Team Foundation Server. HP has developed a Team Foundation Server Bug Item Synchronizer that maintains HP Quality Center bugs synchronized with TFS work items.

These tools were successfully used by a number of TFS customers. One such example is Microsoft's development division that was mirroring the existing defect tracking and source control systems for about two years prior to completing the transition to TFS.

4.2.2.13 Breadth of vendor support

There are a number of products that integrate with VSTS or extend it. As mentioned above, a significant number of smaller vendors exploit VSTS extensibility mechanisms to deliver TFS-based solutions to other operating systems, including mainframe, Linux and real time operating systems. Another important sector of third-party solutions is synchronization tools. A number of large software vendors, such as HP and IBM provide products for bidirectional synchronization with TFS.

4.2.2.14 Success stories

VSTS is publicly available for a number of years and has a wide installation base. The Microsoft Web site publishes about 120 case studies of successful VSTS deployments. These case studies cover customers from a wide spectrum of industries (from aerospace to grocery) and numerous countries (from New Zealand to Russia) and span the period of the last three years, starting at early October 2005. But the most impressive success story of VSTS implementation is its usage by Microsoft. By the end of 2008 most of Microsoft development organization and its internal IT organization use VSTS. The VSTS and the Developer divisions use Team System for all ALM activities it supports. The Office, the Windows and the SQL divisions use some of the VSTS

functionality and are planning to expand its usage. With more than 14,000 users, this is by far the most massive deployment of the VSTS. According to Stephanie Saad (Saad, 2008), the system is extremely popular among its users and gets very high marks from the management team. As one such example she quotes a product unit manager saying "*as we came to recognize the flexibility in the tool, we essentially changed our process on the fly*".

4.3 Comverse's DiME

DiME (Koenig, 2003) is a proprietary integrated and collaborative environment developed and internally used by Comverse. DiME supports a significant part of the product lifecycle from definition to development and delivery. DiME was created as a part of software process improvement effort. Its first deployment took place in 2001 and at that time it was a revolutionary and unique integrated lifecycle management system. Since then DiME has significantly evolved, added more functionality and spread to many Comverse' divisions. By the end of 2007 DiME had more than 1500 active users within Comverse, while adding few hundred each year. DiME provides an integrated management system with support that goes far beyond the pure development activities, offering advanced requirements and release management features. DiME's information model comprises organizational, technical and knowledge data related to the software product lifecycle from definition to delivery and maintenance. DiME is different from other ALM platforms and environments in the following ways:

- It is a proprietary, internally developed and deployed system. DiME is tailored to the existing enterprise product lifecycle, yet possesses flexibility to meet new requirements. Unlike generic commercial tools, DiME had a very detailed features specification. The challenge of meeting such detailed requirements raised the practical quality of the system to a level that is usually not demonstrated by off-the-shelf products.
- Its information model is very broad, incorporating organization hierarchy, customer relations and all data related to product definition, development, delivery and maintenance. Thus DiME bridges between the ALM and ERP domains.
- It is a mature product that is integrated in a large organization for a number of years. Unlike most existing ALM solutions, DiME has a long record of successful large scale organizational implementation.

4.3.1 Design Goals and Architecture

DiME was initiated with a goal of improving existing product development processes that involved a large number of tools, templates, practices and procedures. The process in place suffered from data redundancy and lack of consistency across the organization. The DiME team envisioned a solution for this problem in a form of "ERP for product definition, development and delivery" – a single integrated and collaborative system managing all the data related to these three parts of the lifecycle. The DiME design guidelines included (Koenig, 2003):

- a unified information model that will assure data consistency,
- a central repository holding and managing all the data,
- centralized accessibility to the data,
- needs-based views of the data,
- unified user interface,
- cross-tool workflow automation.

Architecturally DiME is built upon SmarTeam (now ENOVIA SmarTeam) (Dassault Systemes, 2009), a platform for Product Lifecycle Management (PLM) solutions. The platform is responsible for all data management aspects, including data replication and remote access. It also provides some useful functionality such as native UI and document management capabilities.

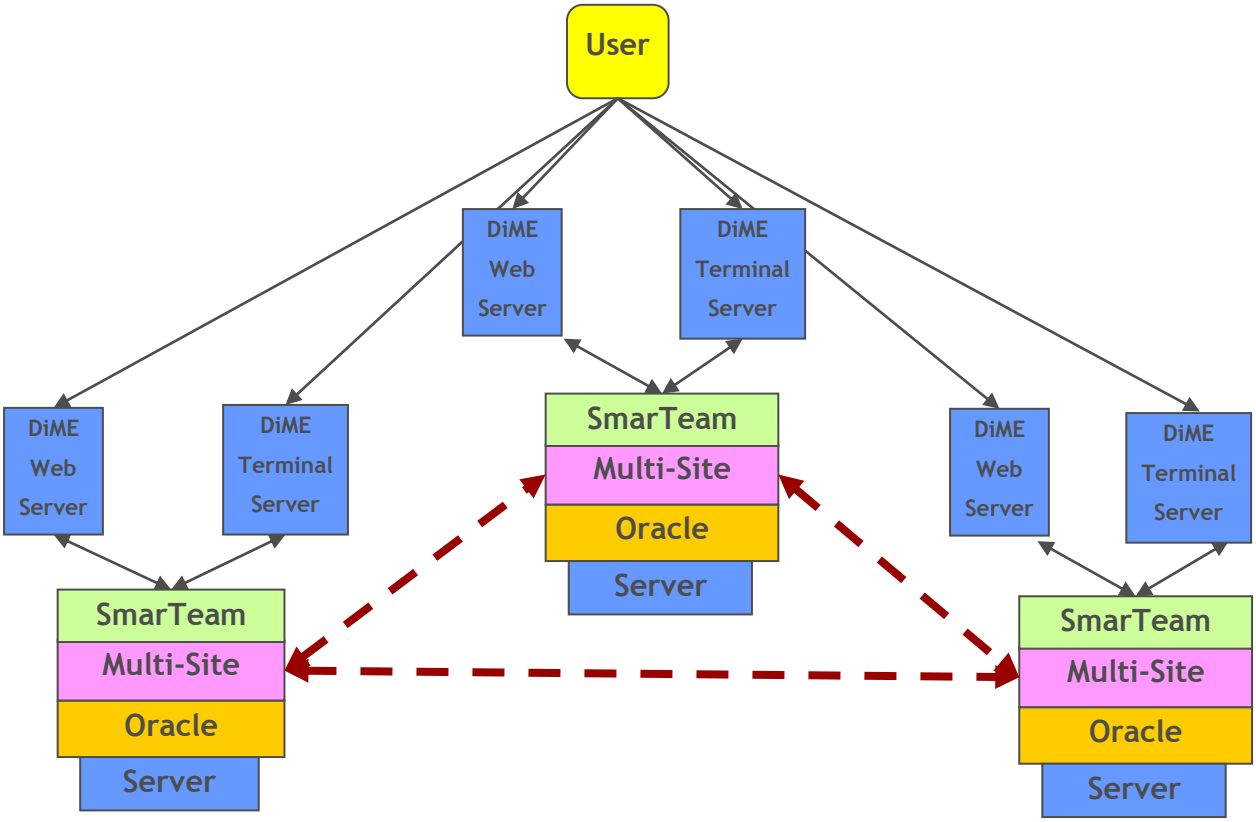

Figure 18. DiME Architecture (from (Koenig, 2008))

DiME's current deployment scheme includes a number of production servers in Israel and the US. The data is replicated between these servers by the underlying Oracle database.

4.3.2 DiME Evaluation

4.3.2.1 Breadth of lifecycle support

DiME offers very broad lifecycle support – the most comprehensive the author has seen so far. Its capabilities include (Koenig, 2008):

- **Product portfolio management features**:
  - **Product tree management.** DiME organizes all products developed by the company in a single hierarchy. The hierarchy consists of products made of subsystems and components, both developed in-house and by third parties. The components can include software and hardware.
- **Project planning and management features**:

- **Release management.** DiME provides powerful release planning, tracking and release management features. A release refers to a specific product or component, may consist of several builds and is linked to a number of development requests. Product releases are managed as hierarchal object and may contain child releases. A release has a lifecycle managed by release manager. DiME supports release planning and tracking by generating release plans and release status reports. The reporting capabilities of DiME will be described further on. DiME provides the **Risk management** features to define, analyze and track the risks that can impact the release. Release may have associated **limitations** and **issues** that are incorporated in the generated documentation and may be inherited between releases.
- **Iteration management.** Releases are composed from builds that incorporate some subset of the release content. Builds are defined during the release planning, then delivered and tested.

- **Product management features**:
  - **Feature/Service management.** Services correspond to the areas of product functionality. DiME maintains a hierarchy of services and features linked to these services.
  - **Requirements management.** Requirements are the core information elements of the DiME's data model and the product lifecycle. They describe the developed system at different levels. Feature requests are translated into feature requirements. Every development request is also accompanied by one or more development requirements. Requirements can be decomposed, have parent-child relationships and linked to related requirements and test cases. DiME provides full traceability between related requirements, test cases, development requests and services. In addition DiME enables automatic generation of numerous requirements specifications and testing documents. Requirements can be maintained in a MS Word document fully synchronized with DiME.
  - **Alarm management.** DiME manages alarms that products issue.
- **Development management features**:

- **Development Requests management.** A Development Request (DR) object can represent any development-related task. Examples of DRs are:
  - new feature implementation,
  - bug fixing,
  - test execution,
  - product deployment,
  - customization implementation,
  - documentation writing.

  DR in DiME is in many ways equivalent to a work item in other systems. DRs are associated with a single Configuration Item (CI) and are committed to a release. A DR can be decomposed into smaller DRs. DRs have a lifecycle, which can be customized to the needs of a specific CI, Release or DR type. DRs are tightly integrated with the related requirements, test cases and release. DiME facilitates DR effort estimation. The estimation is part of the DR and can be used for decision making regarding the DR, for planning, progress tracking and reporting.
- **Defect management.** Defects are development requests that exist in a context of Problem Report. They are linked with features, test cases, iterations and releases.

- **Customer-related management features**
  - **Customer project management.** Customer Project is the main tool for *"defining, analyzing, approving and managing customer commitments and deliveries"* (Koenig, 2003). The customer project management features are mainly used by customer project managers and providers of professional services yet are fully integrated with the rest of the system. Customer Projects are managed by Customer Project Managers that break down the project into Project Work Requests (PWR) and track progress of PWR analysis, estimation and execution.
  - **Customer management.** DiME holds information about the company customers linked to their Customer Projects. This information is synchronized with the ERP system used by the company.
  - **Professional services management –** DiME facilitates professional services activities: customization, deployment, training, documentation, support. All these

can be estimated, planned, tracked and analyzed in DiME with full traceability to all related objects (documents, development requests and requirements).

- **Test management**. DiME supports test cases definition and linkage to development requests and requirements. Test Cases can be grouped into Test Groups, which are part of the Test Specification of some Release. DiME also supports test execution planning. A Test Case execution is Test Instance; Test Instances are executed as a part of Test Session, which belongs to a Test Plan linked to a specific Release. Tests are linked to their related DRs and requirements. The recorded test results are stored in the repository. DiME facilitates test sessions reporting and tracking, creation of bug status reports and other data analysis functionality.
- **User management**. DiME defines and manages a hierarchy of user roles and groups. Roles are associated with activities they perform. Examples of DiME roles are: CI (Configuration Item) Manager, Release Manager, Product Manager and DR Current Responsible. Each role participates in certain lifecycle part of different DiME object types (products, releases and development requests). DiME's user model reflects the actual hierarchy of the organization holding items corresponding to organizational units (business units, departments, teams).
- **Generic capabilities**
    - **Document management.** DiME manages documents as versioned objects that can be linked to other objects in the system. A document can be checked-in and checked-out and possesses an approval lifecycle. This aspect of document lifecycle management together with rich document generation capabilities differentiate DiME from other ALM systems. Both documents revisions and their metadata are stored in DiME's repository.
    - **Knowledge management.** Knowledge items can hold any additional information related to any object in DiME. Examples of knowledge items include: ideas, lessons learned, known issues and news.
    - **Event management.** DiME offers unique and powerful capabilities in managing meetings (events). Meetings are an important part of product management; they accompany almost every part of the lifecycle. Examples of events managed by

DiME are reviews, audits, gate meetings and surveys. The event may have an associated questionnaire (used in gates and reviews). The event usually results in a list of action items that are stored and managed by DiME. Events are linked to their related managed entities, such as DRs, builds and releases.

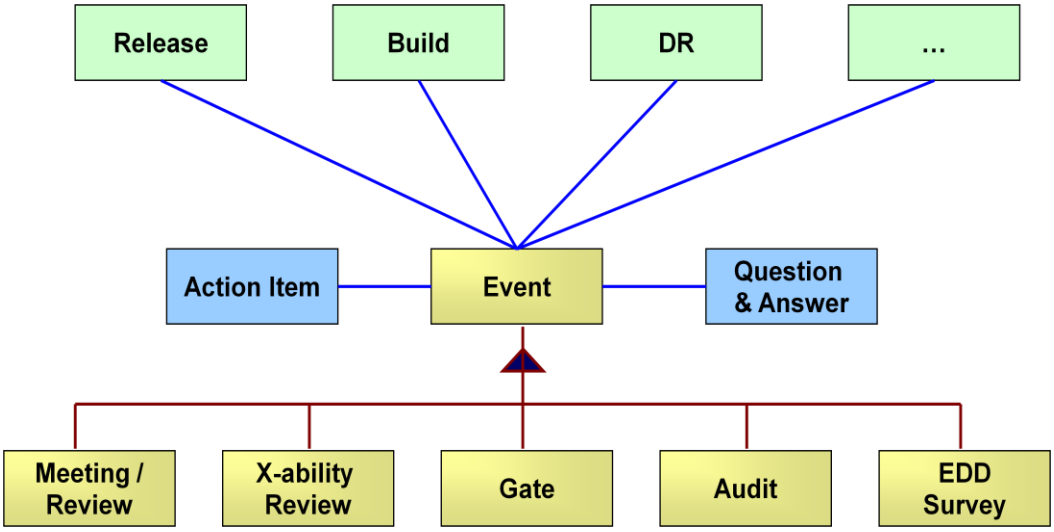

Figure 19. DiME Events

- **Encryption scheme management.** Export regulations impose strict requirements on encryption schemes used by different components of the product. DiME helps managing and retrieving this information.

There are a number of features missing in DiME that are present in many other ALM systems. These include source code version control, build engine integration and build execution.

4.3.2.2 Integration

Platform integration. DiME users access its client application via terminal servers. This solution provides full platform transparency as long as the platform supports remote connection to the terminal server. This approach however requires additional documents copy between user working environment and the terminal server session she is using.

Data integration.

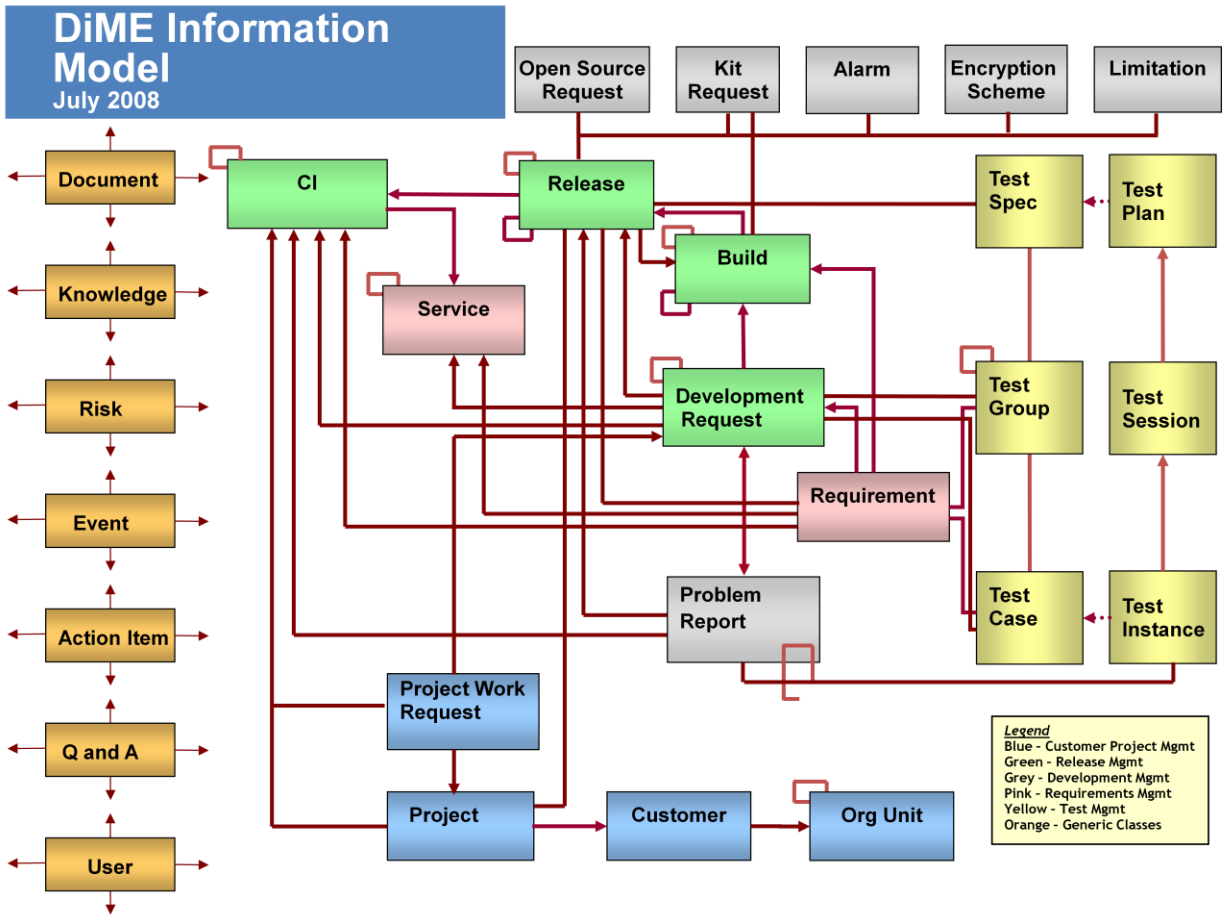

Figure 20. DiME Information Model (from (Koenig, 2008))

DiME is based on an integrated information model that ties together entities from numerous aspects of product lifecycle management. The most important entities are Configuration Item (CI), Release, Development Request (DR) and Requirement. The basic entity type is *DiME Class*. Classes have attributes, can participate in inheritance relationships and can be linked to other classes. The class model is static – it is rigidly defined by the system's developers. Users can't add new data types, attributes or link types between classes. Many DiME classes have a lifecycle – a sequence of states the objects of the class go through. In each state the object is assigned to a responsible user. A special kind of class is *User*. User can belong to one or more *user roles* (e.g., CI Manager, DR Manager and Release Manager). User roles are used by the system for authorization, workflow and automatic notifications. DiME keeps the history of an object's attribute changes.

Presentation integration. DiME has a Windows GUI interface that exposes all its features. No other presentation integration issues are relevant for it.

Process integration. DiME supports workflows by implementing a notion of lifecycle. Many DiME classes have a predefined lifecycle. The lifecycle consists of states that a class' objects go through. Each state has an associated responsible and fields marking the time of entering and leaving the state. Transition from state to state is accompanied by email notifications to relevant users. Transitions are subject to conditions: for example a user can not promote a DR to the integration state if the DR is not yet committed to a release.

4.3.2.3 Role-based views

While DiME does not offer different functionality subset as a function of the user role, it has effective means to address this issue. Two aspects of DiME user interface perform the functions of the role-based views. DiME has a Web interface that exposes a subset of DiME functionality relevant mostly to the customer-related capabilities. The Web portal integrates DiME with other features backed by Comverse ERP and other systems.

The second such aspect is implemented in a form of task-based wizards. Every screen of DiME user interface offers an easy access (using a so-called "rainbow button") to wizards for most popular tasks related to that screen. This complements the traditional data-centric user interface that DiME inherits from its SmarTeam platform.

4.3.2.4 Traceability and reporting

DiME exposes rich reporting capabilities. Users can generate new reports on data stored in the repository using SmarTeam and SQL queries. These queries allow complex conditions and filtering expressions. The system also provides shortcuts to many predefined reports that are automatically generated in a number of widespread formats, such as MS Excel. Reports serve as a convenient front end for many planning and traceability features of DiME. For example, the DR Test Execution Report informs about all requirements linked to the DR, all test cases linked to these requirements and the results of these test cases executions. The Requirements Impact Analysis Report brings together all objects linked to the chosen requirement: requirements, DRs

and test cases. The generated graphs and reports are the main tools for development iterations and test execution planning and control. One of the most important tools for tracking the release progress is the Earned Value Chart automatically generated by DiME. DiME saves the daily earned value points thus allowing to assess release development velocity and to forecast the code freeze date.

4.3.2.5 Process definition and automation

DiME's process automation support is based on the abovementioned lifecycle of DiME classes. It is worth noting that the class' lifecycle cannot be changed by the user – the user can not add new states. They are rigidly defined by the system developers. However, the user can mark some parts of the lifecycle as not relevant. The DiME's process support could be described as semi-automatic, there is no programmatic way to trigger state transitions, to specify checks to be performed prior to state change and to define actions that must happen upon such transitions.

4.3.2.6 Platforms support

DiME client is a Windows application. Currently DiME does not have clients running on other operating systems. However DiME efficiently solves this problem by exposing access to it client application via terminal servers. These servers can be accessed from any platform that has a remote terminal access.

4.3.2.7 Extensibility and openness

DiME is a proprietary system and thus is not meant to be extended by third parties.

4.3.2.8 Distributed teams support

DiME's distributed development support consists of two aspects. The first aspect is repository mirroring that is provided by the underlying SmarTeam platform. The second aspect is accessing DiME by remote users that is achieved by Microsoft Remote Desktop Connection Servers. Thus remote teams can work with the geographically closest mirror of the data. Currently teams from India, China, Ukraine, Australia, Israel, the UK and the US are actively using DiME.

4.3.2.9 Scalability

DiME scalability is provided by the underlying SmarTeam platform. It effectively supports hundreds of simultaneous users without noticeable performance degradation. The scalability tests conducted by the development team discovered that the performance bottleneck is the amount of physical memory of the terminal servers.

4.3.2.10 Security

DiME uses SmarTeam access authorization mechanism and manages user authorization and groups. Its Access Authorization subsystem uses complex algorithms to calculate the entities each user group is allowed to access based on internal policies and the current group responsibilities.

DiME currently lacks integration with LDAP and Active Directory thus forcing users to log in twice.

4.3.2.11 Incremental implementation

DiME's deployment in Comverse was incremental, by modules (Koenig, 2003). First, the development divisions installed the development core module, and then the customer project module was installed by sales and professional services. More modules (Requirements Management, Document Management and Testing Management) were added gradually to the installation.

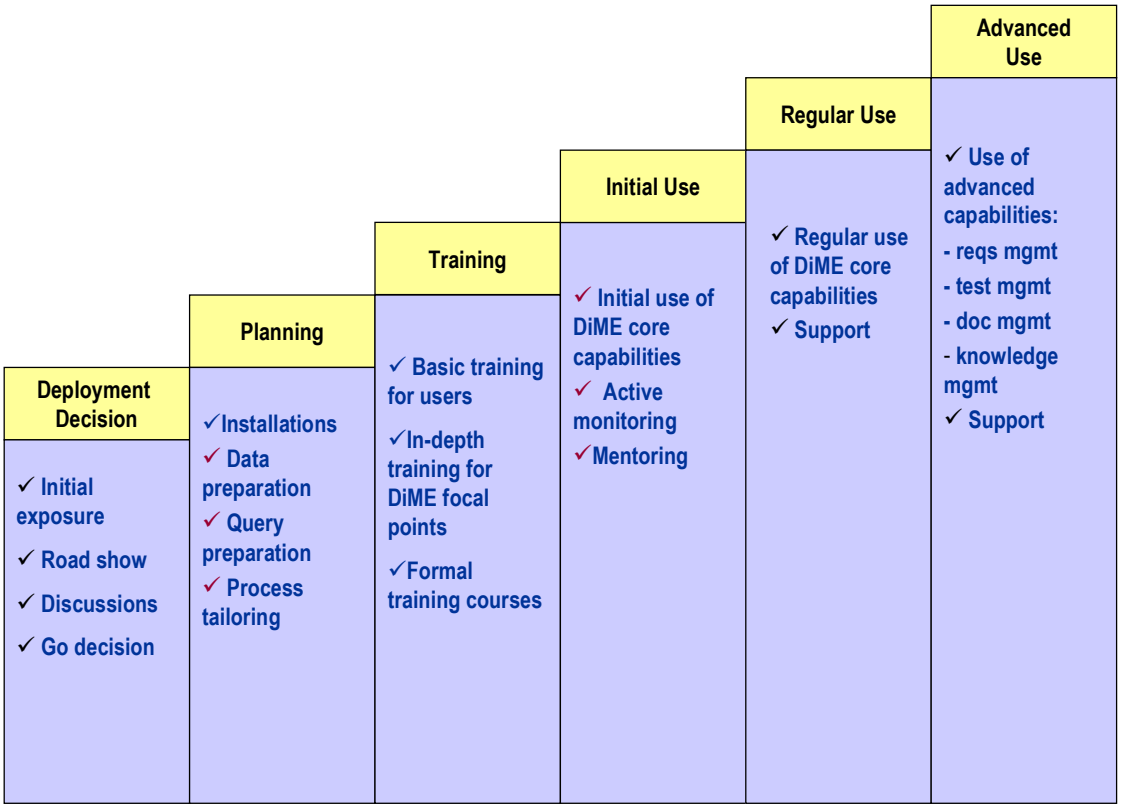

Figure 21. The DiME Deployment Process (from (Koenig, 2003))

The DiME development team invested a lot of thought, efforts and careful planning in the system implementation. These investments have paid back – the number of active DiME users grows from year to year and has topped 1500 in the 2007. The number of active DiME objects surpassed 400,000 in the same year (Koenig, 2008).

4.3.2.12 Integration and interoperability with existing solutions

DiME has interfaces to Rational ClearQuest defect tracking system, Harvest source control system, HP Quality Center, Oracle Applications ERP system and with a Comverse internal purchase order management system.

4.3.2.13 Breadth of vendor support

This subject is not relevant to DiME.

4.3.2.14 Success stories

The successful implementation of DiME inside Comverse is the best witness of its quality.

4.4 Summary

In order to present the study of the ALM solutions in a short and concise form we will bring together the evaluation of each tool by the classification aspects. Each aspect will get a mark of:

- Needs improvement
- Average
- Strong

This ranking is very approximate, rough and can't reflect the details and specifics of each aspect. These details are covered in depth in the content of the corresponding section of this paper. However having all aspects summarized in one table can provide a better overall picture.

Aspect/Product	Jazz	Team Concert	VSTS	DiME
Breadth of lifecycle support	N/A	Average	Average	Strong
Platform integration	Average	Average (based on Jazz)	Strong	Needs improvement
Data integration	Strong	Strong (Open Services)	Strong	Strong
Control integration	Strong	Strong (based on Jazz)	Strong	N/A (implemented as a monolithic system)
Presentation integration	Strong	Strong (based on Jazz)	Strong	Average
Process integration	Strong	Strong (based on Jazz)	Needs improvement	Average (predefined lifecycle)
Role-based	N/A	Needs	Strong	Average

views		improvement		
Traceability and reporting	Strong	Strong	Strong	Strong
Process definition and automation	Strong	Strong (based on Jazz)	Needs improvement	Average (predefined lifecycle)
Platforms support	Average	Average (based on Jazz)	Strong	Average (though terminal servers)
Extensibility and openness	Average	Strong (Open Services)	Strong	N/A
Distributed teams support	Strong	Strong (based on Jazz)	Strong	Average
Scalability	Average	Average (based on Jazz)	Strong	Average
Security	Average	Average (based on Jazz)	Strong	Average (proprietary)
Incremental implementation	Strong	Needs improvement	Needs improvement	Strong
Interoperability	Average	Average	Average	Average
Breadth of vendor support	Needs improvement	Needs improvement	Average	N/A
Success stories	Needs improvement (still new)	Needs improvement (still new)	Strong	Strong

5 Trends and tendencies of ALME`s

This section summarizes and generalizes the trends in ALME development. It represents the author's opinions and conclusions made after studying the past and the present of the field. These opinions have been formed by reading academic and analysts' researches, by analyzing the technical aspects of existing systems, and by numerous discussions both written and oral with specialists involved with ALM.

One can see more and more interest and activity in the ALM domain and it seems that this tendency will only grow. Today's software development management requires integrated management systems. Individual tools can't cope effectively with the complexity, the large scale and the distributed nature of software development. Because of the problem's difficulty, the solutions are also complex. Software solutions providers approach this complexity in different ways, depending on their size and existing portfolio.

5.1 Large ALM software vendors

A number of the largest software vendors have already stepped into the ALME market and provide integrated solutions. The clear leaders are Microsoft with VSTS and IBM with its Jazz Platform and Team Concert suite. Many others, such as HP and MKS, have declared about creation of an integrated ALM solution. Below are some common observations for this vendors' sector.

- **Building from scratch vs. extending existing products.** Trying to tie different tools together is exactly the point of failure of ALM 1.0 systems. The proper solution for the integrated ALM should be designed and built from the ground up. However the cost of building such complex system from scratch is very high. It seems that only the largest companies can afford it. This is the approach both IBM and Microsoft have chosen. The author doubts that in the near future many companies will follow them. Still, those who can afford proper design will eventually offer the best proposition. An alternative is to redesign existing systems so they can be tied together better. It seems that a number of vendors will choose this option because it can result in much shorter time-to-market.

- **Incremental functionality growth.** Practically all software vendors have realized that it is impractical to build a system that will provide all possible functionalities at once. Instead, they choose a small number of core features to be delivered first and to serve as a basis for future additions. This process characterizes both Jazz and VSTS. The core set of features is usually based on the activities where the vendor has most competence. Thus IBM's Team Concert first components were the source control and work items management. This might be related to a lot of experience that Rational gained with its ClearCase and ClearQuest products. Microsoft put a lot of effort into coding support features of VSTS. HP's Project and Portfolio Management Center inherits a lot of experience from its QualityCenter product. Once the ALM solution platform is in place, the platform's vendor adds more features incrementally, while keeping them tightly integrated.
- **Single platform consolidation.** Another important trend in this sector is consolidating the vendor's products around the ALM platform. The products that can't be integrated with the platform are considered legacy. For example, IBM has declared that its leading configuration management product, ClearCase, will not be further developed. Instead, the existing customers will be migrating to a Jazz-based configuration management. The author assumes, that this trend will continue and vendors will try to integrate as many of their products as possible under a single umbrella. This will both bring maximal value to the customer and decrease the vendor's maintenance expenses.
- **Rich extensibility features.** Large vendors see a lot of value in extension capabilities of their systems. They prefer to give others powerful options of integrating into their platform instead of founding their platform on a common standard. Thus they keep their market dominance and can even extend it. This is probably the very reason there is no common standard in the ALM domain. One can see this behavior both with Jazz and VSTS. One particularly interesting example is the story of the ALF (Application Lifecycle Framework) project (Eclipse Application Lifecycle Framework (ALF) Project) that aimed to provide an Eclipse-based standard for development tools interoperability. ALF is an open source project that was supported by a group of software providers such as Serena Software, Catalyst Systems, Compuware and Segue. However, IBM, which

was an initiator of Eclipse and still is one of the major factors behind it, preferred to stay away from this initiative and developed its Eclipse-based Jazz platform. Likewise, other major vendors (Microsoft, HP, MKS and Oracle) have ignored this initiative and now it seems to be virtually dead. At the same time large vendors invest a lot of thought and effort in their systems' extensibility and conduct special partners programs for the companies that seek to create such extensions. The author believes this tendency will strengthen.

5.2 Medium and small vendors

Medium-sized and small software companies pursue a different strategy for creating ALM support systems. Because creating a complete ALME supporting multiple activities is usually too costly and has low chances of winning the market, they prefer the following strategies:

1. **Grouping together with other similar solution providers.** The goal is establishing a common standard platform with each small vendor contributing some lifecycle support tools based on this platform. One example of such effort is the ALF project mentioned above. This strategy is problematic in practice. As we've seen in the case of IPSE, it is very hard to agree and establish a common set of interfaces and infrastructures so that it satisfies many parties with different needs and priorities. Such infrastructure implementation is both expensive and time consuming. Unless there is one leading vendor that governs the joint effort the author believes this strategy will not reach any practical success. This happened with the ALF project, which was practically abandoned after failing to gain large vendors' support.
2. **Extending or complementing larger vendors' solution.** Moving their products to established ALM platforms and filling the functionality gaps seems to be an option for smaller vendors. Establishing a partnership with large vendors can expose them to wider market relatively easy. The author thinks that we will see more products migrating to IBM's Jazz and Microsoft's TFS. A number of products which provide functionality missing in Team Concert and VSTS already exist.

3. **Supplying an integration platform to bind other vendors' products.** The vendors that take this approach build the ALM engine and adaptors to existing tools. The engine provides process automation, data synchronization, data integration and reporting across the tools. The adaptors connect the tools to the engine, probably by means of some common integration hub. In addition, the engine provider can supply its own tools for some ALM activities. This scheme allows the organization to keep its existing tools and practices, preserves the value invested in them and adds new capabilities that result in integrated and automated lifecycle. The strategy is appropriate for consulting companies that can customize the existing engine and adaptors to the specific customer configuration. One of such providers is Kovair (Kovair). This direction seems promising, and its potential will grow with utilization of new open source technologies and new ALM features of the tools. However, binding together separate tools is a very challenging task and the quality of the tools integration will never reach the level of the systems originally designed for the integrated lifecycle. The complexity of building, maintaining and continuous update of multiple adaptors questions the scalability of this approach. Another problematic aspect of this solution is a lack of common user interface to the multiple tools that it binds together.

5.3 New ALM solution providers

Building an ALM platform from zero is expensive and the ALM products market is already saturated with mature products with numerous customers. Therefore, it is unlikely that many new companies will try to enter this market. One potential option for a new player aiming to create a complete ALME solution is to acquire some existing product with significant market share. The author expects that this strategy will be undertaken by large companies with products in the disciplines adjacent to ALM: ERP, IT management and Operations management. Oracle is the first such company trying to enter the ALM market without

having existing products for development support. It has declared this intention in 2007 (Oracle Eyes the ALM Market , 2007). The three main elements of the Oracle's strategy are:

1. **Connecting product development with the business process.** This will help Oracle to expand its presence into the development sector of organization.
2. **Focusing on interoperability with other providers' tools.**
3. **Connecting product development with operations.** This is another aspect of the lifecycle Oracle is strong at.

Oracle plans to build its ALM proposition with the help of acquisition of companies having experience in this domain. A recent acquisition of Primavera, a large provider of project and portfolio management solutions, is another step in this direction. However, there is some skepticism among analysts regarding Oracle's ability to provide a solution for non-Oracle applications (Oracle Eyes the ALM Market , 2007).

Another interesting direction is the converging of ALM and PLM (Product Lifecycle Management) domains. PLM solutions manage all product related data in non-software industries, such as automotive, semiconductor and others. Software product lifecycle has a lot of common with the more generic product lifecycle managed by the PLM systems. The author expects (based on these domains convergence) that some PLM vendors will try to expand their offering to the ALM domain. Alternatively, PLM platforms can be utilized to build ALM solutions (similar to DiME). The latest trend in the PLM products is called PLM 2.0 (Dassault Systemes). It adds Web 2.0 concepts to PLM solution: Web-based online access, online collaboration and social community capabilities.

5.4 Open source and ALM

Open source community is a significant factor in the development world today. Open source tools are very popular and represent a large market share. ALME`s involve open source in a number of ways:

- **Commercial products that open part of their source code and database schemas.** Jazz is one such example. This strategy attracts developers outside of the product vendor

to contribute to the product by creating extensions, exploring the new features and finding bugs.

- **ALM products that interoperate with open source systems.** This is a very common practice and many ALM systems already can integrate with popular open source tools (e.g., Team Concert Subversion integration) or can employ open source components in their architecture (e.g., Jazz using Apache Web Server).
- **ALME composed from open source components.** A lot of lifecycle management functionality and infrastructure services are available as open source products. It is possible to combine these tools into an integrated ALM solution. A number of such solutions exist. They are built by consulting companies that put separate tools together, manage the connections between them and tailor the solution to the customer. Polarion (Polarion® ALM — Everthing you need in one single ALM solution) brings together Subversion version control, Apache Web server, Open API for Website interactions, OpenSymphony for workflow management and the Eclipse IDE. TeamALM by nexB (nexB, 2008) also follows this strategy. This approach appeals in its openness, utilization of existing products and non-dependency on single vendor. On the other hand, the possibility to integrate separate products not initially built for such integration seems very questionable. This was the weak point of ALM 1.0, and open source products will encounter the same problems as the commercial ones. Another potential problem is dependency on many products without real ability to influence their development.

5.5 Proprietary ALM solutions

While there are a lot of vendors offering products for integrated lifecycle management, there is still a place for proprietary systems. The following reasons can motivate an organization to create a proprietary ALM system:

- Existence of an established, well-understood lifecycle management process,
- Unique needs or specific development or operational environment,
- Reluctance to open up its internal business, development and organizational information to external consultants or vendors,

- Existence of internal competence and experience in building the required solution.

Developing the ALM solution in-house can result in a better match to the organization's needs, lesser dependency on outside factors and smoother system adoption. Reusing existing infrastructure and open source components can accelerate the proprietary systems development. At the same time, creating and maintaining a complex ALM environment is a large task that will consume significant amount of resources. Another argument against developing ALM solution in-house is a tendency of tools standardization, which is especially strong among large enterprises.

5.6 Technology trends of ALME`s

Existing ALM products are built using multiple and diverse technologies. Still we can highlight a number of technological trends.

- **Reuse of existing technological components.** This seems to be a major element in the architecture of today's and tomorrow's ALM systems. Today's software offers a lot of reliable, powerful and generic ready solutions for many infrastructure aspects of a software product. These aspects include data management, Web interface, communication, user interfaces and many others. Composing a solution from proven and standard building blocks allows concentrating on the essential functionality the system should provide.
- **Competition between the single-repository and the repository-neutral architectures**. Both architectures are employed in existing systems and both are legitimate. However, it seems that the flexibility of the repository-neutral approach is a key advantage. While there is a cost to complexity of managing the data synchronization and integration, a properly designed ALM platform will hide this complexity from the applications built upon it.
- **Web service interfaces.** Web services became a central element of the ALM systems integration. Standard and simple technology, reliable infrastructures, excellent scalability, security, distributed support, ease of access and platform compatibility make Web services a natural choice for the primary communication technology for an ALM

platform. The author believes this trend will become even more dominant together with SOA (Service Oriented Architecture).

- **Integration of communication and collaboration features.** A number of ALME`s are characterized by tight integration of the popular communication and collaboration technologies. Making instant messaging, chat and RSS feed part of the developer's working environment facilitates better communication, makes distributed development easier and more productive and fits well with modern Agile development practices.
- **Process automation and governance.** This aspect is one of the ALM pillars and it should become even more important as ALME`s span to include the operation, IT and business parts of the organization. Another factor that influences the importance of process automation is the process standardization trend of large organizations, especially a wide acceptance of ITIL (Information Technology Infrastructure Library) (ITIL® Home, 2009). ITIL brings together a set of best practices for IT Service Management. ITIL has been adopted by hundreds of large organizations worldwide, including Microsoft, IBM, HP, HSBC and many others. The discussion of ITIL and its relationship to ALM goes beyond the scope of this work.
- **Raising importance of reporting capabilities.** In today's software development with its large projects developed by globally distributed teams, knowing the state of the project at every moment of time is as crucial, as it is difficult. The ALM solution must bring accurate and detailed real-time project state picture that combines data from multiple tools. These abilities become even more important as ALM systems start serving organization's upper management. The author expects to see a lot of effort invested in advanced reporting and traceability features of ALME`s, including data analysis and data mining technologies. One of the first implementations of such advanced data analytics functionality is Orcanos's Business Intelligence for ALM 2.0 infrastructure (OASIS) (Orcanos, 2008). OASIS features include real-time dashboards, metrics-based alerts (based on project data – slow progress, low quality, etc.) and decision-making tools. While the technology itself seems not mature enough to be a reliable decision-making enabler, it definitely is very promising and the author expects more ALM platforms to deliver such functionality.

- **Role-based views.** Integrated Development Environments (IDE`s) were the first tools powered by ALM platforms. Having the development activities as the first supported ones, ALM platforms gradually widen their lifecycle support to other parts of the organization – product and project management, upper management, IT and operations. This calls for creating role-specific views into the data managed by an ALME. As the user audience of ALME`s widens and becomes more heterogeneous, we should see the increasing importance of the tools and technologies providing role-based views. Tomorrow's ALM environments will have to present their data in the format that meets the user's expectations, needs and experience. It is also expected more functionality will become accessible by a light Web interface that requires no specialized software installation on the client machine and allows accessing the required features regardless of the client operating system and physical location.

There are many more aspects and directions in this dynamic area of software industry. The trends listed above are the most important ones the author sees in the ALME space.

6 Conclusions

The work started from a systematic study of application life cycle. The goal of this study was to establish terminology used throughout this work and to create an accurate and comprehensive definition of Application Lifecycle Management and environments for ALM. A number of existing definitions were examined and found to be inconsistent. The definition of ALM was divided into the following aspects: category, scope, goals, means, managed entities. Using this methodology the definition of ALM was formulated. This definition unifies the existing definitions and expresses the common shared vision of ALM.

Based on this understanding of ALM, the study of ALME history was approached. The evolution of CASE tools was reviewed devoting special attention to the factors driving this evolution. Two initiatives representing the early attempts to create integrated software engineering environments were studied: IPSE and PCTE. A special focus was placed on the ideas behind the initiatives, their achievements and the reasons for their failure. These mistakes provide important lessons helping to foresee the prospects of current ALM systems. The mistakes included: excessive focus on technology combined with little attention to real world user needs, lack of software process understanding, overlooking the organizational and business aspects and trying to solve the problem too big for existing technology and process maturity. Lastly ALM 1.0 was reviewed – the approach that tries to reach the ALM goals by tool-to-tool integration.

Armed with the terminology, the understanding and the definition of ALM and with the lessons learnt from the mistakes of past initiatives, the author approached the definition of the ALM classification framework. The developed framework took into account the central aspects of ALM support as identified in the previous parts of this work. These aspects included: different kinds of integration and its depth, correspondence to the defined ALM goals and a number of other technical and non-technical characteristics. The author believes that the developed framework possesses breadth, accuracy and the right level of details to allow classification of ALM environments. To the best of the author’s knowledge, no other ALME classification framework exists currently.

The developed classification framework was applied to a number of ALM environments and platforms. The chosen platforms and environments (IBM Jazz and Team Concert, Microsoft Team Foundation Server and Visual Studio Team System and Comverse DiME) represent the leading directions in the ALM space today. These systems demonstrate breadth of support, technological and process maturity, scalability, reliability and practical quality fitting the requirements of large enterprise organizations. Each of the systems has proved itself in significant success stories. The application of the framework to the chosen products helped highlighting their architectural and technological patterns and strategies, their strengths and deficiencies, and trends that will prevail in tomorrow's ALM domain. This framework usage has proven its quality and usefulness.

There are a number of topics relevant to the ALM space and ALME`s that were not covered in this paper. Among them are PLM and ERP, their evolution and relationship with ALM. ITIL, its Application Management part and ASL (Application Services Library) (Pols, 2005), their meaning and influence on ALM were also not covered. A number of important leading ALM solutions, were not classified using the framework. It is possible that adding more systems to the classification would refine and enrich the framework. All these issues can be a basis for future work.

In summary, the domain of integrated environments for ALM is growing and progressing rapidly. All necessary ingredients for this growth are present: the software business became large enough and complex enough to demand integrated solution; the technology is mature enough to deliver such solutions with the required functionality, stability and scalability; finally, the software process itself is mature enough, stable enough and understood well enough. One can see integrated ALM platforms already being deployed and successfully exploited today to deliver extremely large and complex software products. The author believes that in the near future non-integrated ALM tools will be considered legacy and all significant application lifecycle support products will become part of integrated systems. After that, the author expects to see fusion between PLM, ERP and ALM systems into a single management platform that governs,

automates and controls all aspects of an organization in a centralized integrated manner. Looking at DiME one can think that this vision is not that far off.

Appendix 1: List of Abbreviations

ALF Application Lifecycle Framework

ALM Application Lifecycle Management

ALME Application Lifecycle Management Environment

API Application Programming Interface

APSE Ada Programming Support Environment

ASL Application Services Library

CASE Computer Aided Software Engineering

CI Configuration Item

DR Development Request

ECMA European Computers Manufacturers Association

EMF Eclipse Modeling Framework

ERP Enterprise Resource Planning

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

IPSE Integrated Programming Support Environment

ISO International Standards Organization

IT Information Technology Infrastructure Library

ITIL Information Technology

LDAP Lightweight Data Access Protocol

MS Microsoft

MSDN Microsoft Developer Network

NIST National Institute of Standards and Technology

OMS Object Management System

OS Operating System

PCTE Common Tool Environment

PLM Product Lifecycle Management

REST Representational State Transfer

RM Reference Model

RSS	Really Simple Syndication
SEE	Software Engineering Environment
SOA	Service Oriented Architecture
TFS	Team Foundation Server
UI	User Interface
URI	Universal Resource Identifier
VPN	Virtual Private Network
VSTS	Visual Studio Team System
XML	Extensible Markup Language

Bibliography

Abramovici, M., & Sieg, O. (2002). *Status and Development Trends of Product Lifecycle Management Systems.* Ruhr-University, Chair of IT in Mechanical Engineering, Bochum.

Ada Joint Program Office, United States Department of Defense. (1986). *Requirements and Design Criteria for the Common APSE Interface Set.*

Boehm, B. (2006). A View of 20th and 21st Century Software Engineering. *28th international Conference on Software Engineering*, (pp. 12-29). Shanghai, China.

Borland. (2007, April). *Open Application Lifecycle Management (ALM).* Retrieved January 31, 2009, from Borland: http://www.borland.com/resources/en/pdf/company/open-alm-whitepaper.pdf

Brown, A. (1993). An Examination of the Current State of IPSE Technology. *The 15th international conference on Software Engineering*, (pp. 338-347). Baltimore, Maryland, United States.

Brown, A., & McDermid, J. A. (1992, March). Learning from IPSE's Mistakes. *IEEE Software* , 23-28.

Buxton, J. (1980). *Requirements for Ada Programming Support Environments – Stoneman.* U.S. Department of Defense.

European Computer Manufacturers Association. (1990). *Portable Common Tool Environment (PCTE): Abstract Specification, ECMA-149.*

Forte, G. M. (1991). *CASE Outlook: Guide to Products and Services.* Lake Oswego, Oregon: CASE Consulting Group.

Fuggetta, A. (1993). A Classification of CASE technology. *Computer , 26* (12), 25-38.

Fuggetta, A. (2000). Software process: a roadmap. *Conference on The Future of Software Engineering*, (pp. 25-34). Limerick, Ireland.

Humphrey, W. (1989). *Managing the Software Process.* Addison-Wesley.

IBM. (2008). *Collaborative Application Lifecycle Management with Rational Products.* Retrieved July 18, 2008, from IBM: http://www.redbooks.ibm.com/redpieces/pdfs/sg247622.pdf

International Standard Organization (ISO/IEC). (1995). *Information technology - Software life cycle processes.* International Standard, Geneva, Switzerland.

Koenig, S. (2003). Integrated Process and Knowledge Management for Product Definition, Development and Delivery. *International Conference on Software - Science Technology and Engineering.* IEEE.

Koenig, S. (2008). *Introduction to DiME – Vision, Principles and Basic Concepts.* Comverse.

Lewis, G. A., Morris, E., Simanta, S., & Wrage, L. (2008). Why Standards Are Not Enough To Guarantee End-to-End Interoperability. *Seventh International Conference on Composition-Based Software Systems* (pp. 164-173). IEEE.

Long, F., & Morris, E. (1993). *An Overview of PCTE: A Basis for Portable Common Tool Environment.*

Meier, J., Taylor, J., Bansode, P., Mackman, A., & Jones, K. (2007, September). *Chapter 17 – Providing Internet Access to Team Foundation Server.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/bb668967.aspx

Meier, J., Taylor, J., Mackman, A., Bansode, P., & Jones, K. (2008, October 15). *patterns & practices: Team Development with Visual Studio Team Foundation Server.* Retrieved February 1, 2009, from CodePlex - Open Source Project Hosting: http://www.codeplex.com/TFSGuide

Microsoft. (2007, August). *Driving Your Business Forward with Application Life-cycle Management (ALM).* Retrieved January 31, 2009, from Microsoft: http://download.microsoft.com/download/c/2/8/c28f6cef-ff0e-461b-88c0-c93a30f3f67b/78249_ALMwpaper_r3t1_mg.pdf

Morgan, D. (1987, April). The Imminent IPSE. *Datamation* , pp. 6--68.

NIST/ECMA. (1993). *Reference Model for Frameworks for Software Engineering Environments.* NIST Special Publication 500-201.

Norman, R., & Chen, M. (1992). Working Together to Integrate CASE. *IEEE Software , 9*, 12-16.

Oliver, D. W. (1994). *A System Engineering Tool Taxonomy.* Retrieved January 31, 2009, from http://members.tripod.com/Rick_Steiner/ToolTax.pdf

Pols, R. v. (2005). *Application Services Library (ASL): A Framework for Application Management* . Van Haren Publishing.

Rotibi, O. (2006). *Application Lifecycle Market Analysis.* Ovum.

Schwaber, C. (2006). *The Changing Face of Application Lifecycle Management.* Forrester Research Inc.

Shaw, K. A. (2007, April). *Application Lifecycle Management for the Enterprise.* Retrieved January 31, 2009, from Serena Software: http://www.serena.com/docs/repository/company/serena_alm_2.0_for_t.pdf

Singh, R. (1998). *International Standard ISO/IEC 12207 Software Life Cycle Processes.* Washington: Federal Aviation Administration.

Sommerville, I. (2004). *Software Engineering.* Addison-Wesley.

Wasserman, A. (1990). Tool Integration in Software Engineering Environments. *International Workshop on Environments* (pp. 137-149). Berlin: Springer-Verlag.

Web Bibliography

Apache. (n.d.). *Apache Ant - Welcome*. Retrieved July 2, 2009, from http://ant.apache.org/

Borland. (2007, July 28). *Borland Customer Survey Examines the Current State of the Application Lifecycle Management Market.* Retrieved January 31, 2009, from Borland: http://www.borland.com/us/company/news/press_releases/2007/08_28_07_borland_customer_survey.html

Borland. (n.d.). *Software Solutions for Change Management, Asset Management, Test Automation, SDLC and more*. Retrieved January 31, 2009, from Borland: http://www.borland.com/

Compuware Corporation. (n.d.). *Compuware's Quality Solutions*. Retrieved January 31, 2009, from Compuware Corporation Homepage: http://www.compuware.com/solutions/3596_ENG_HTML.htm

Dassault Systemes. (n.d.). *Dassault Systemes.* Retrieved February 1, 2009, from Bringing PLM 2.0 to Life: http://www.3ds.com/fileadmin/PRODUCTS/ENOVIA/PDF/V6Brochure-0807_hq__FINAL.pdf

Dassault Systemes. (2009). *ENOVIA SmarTeam Engineering Express.* Retrieved February 1, 2009, from Dassault Systemes: Product Lifecycle Management PLM and 3D Simulation Software Solutions: http://www.3ds.com/products/enovia/mid-market/smarteam-engineering-express/overview/

Eclipse. (n.d.). *Eclipse Application Lifecycle Framework (ALF) Project.* Retrieved February 1, 2009, from Eclipse.org home: http://www.eclipse.org/alf/

Eclipse. (2005, June 16). *The Eclipse Modeling Framework (EMF) Overview.* Retrieved February 1, 2009, from Help - Eclipse SDK: http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html

Harry, B. (2007, October 18). *TFS 2008 System Recommendations.* Retrieved February 1, 2009, from bharry's Weblog: http://blogs.msdn.com/bharry/archive/2007/10/18/tfs-2008-system-recommendations.aspx

IBM. (n.d.). *Jazz Overview*. Retrieved January 31, 2009, from IBM: http://www-306.ibm.com/software/rational/jazz/

IBM. (2008, June 24). *Jazz Platform Technical Overview.* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/learn/LearnItem.jsp?href=content/docs/platform-overview/index.html

IBM. (n.d.). *Open Services for Lifecycle Collaboration.* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/open-services/index.jsp

IBM Rational ClearCase V7.1. (n.d.). Retrieved July 2, 2009, from http://www-01.ibm.com/software/awdtools/clearcase/

IBM Rational ClearQuest V7.1. (n.d.). Retrieved July 02, 2009, from http://www-01.ibm.com/software/awdtools/clearquest/

IBM. (n.d.). *Rational Team Concert Capabilities.* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/learn/LearnItem.jsp?href=content/docs/rtc1.0-capabilities/index.html

IBM. (n.d.). *Typical Lifecycle Resources and Relationships.* Retrieved February 1, 2009, from Jazz Community site: https://jazz.net/open-services/resources/alm-resources.jsp

ITIL® Home. (2009). Retrieved February 1, 2009, from http://www.itil-officialsite.com/home/home.asp

Jonston, S. (2008, September 6). *JRS has graduated!* Retrieved February 1, 2009, from IBM developerWorks:Blogs:Tooling platforms and RESTful ramblings: http://www.ibm.com/developerworks/blogs/page/johnston?entry=jrs_has_graduated

Kovair. (n.d.). Retrieved February 1, 2009, from Kovair Global Software Development Lifecycle (SDLC) Management: http://www.kovair.com/

Lainhart, T. (2008, March 13). *The Jazz Feed Service.* Retrieved February 1, 2009, from Jazz Team Wiki: https://jazz.net/wiki/bin/view/Main/FeedService

Lemieux, J.-M. (2008, November 25). *Performance and Scalability.* Retrieved February 1, 2009, from Jazz Team Wiki: https://jazz.net/wiki/bin/view/Main/PerformanceScalability

Lemieux, J.-M. (2008, July 9). *Self-hosting by the numbers - June 2008.* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/blog/index.php/2008/07/09/self-hosting-by-the-numbers-june-2008/

Lemieux, J.-M. (2008, October 1). *Towards a Visual Studio client.* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/blog/index.php/2008/10/01/towards-a-visual-studio-client

Microsoft. (2009). *About Visual Studio Team System.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/vsts2008/products/bb964615.aspx

Microsoft. (2009). *Compare MSDN and Expression Subscriptions.* Retrieved February 1, 2009, from MSDN: Microsoft Development Network: http://msdn.microsoft.com/en-us/subscriptions/subscriptionschart.aspx

Microsoft. (2008). *Cross-Platform Developmet Solutions.* Retrieved December 18, 2008, from Microsoft: http://msdn.microsoft.com/en-us/vstudio/products/cc197931.aspx

Microsoft. (2005, October 25). *Microsoft Case Studies: HP.* Retrieved February 1, 2009, from Microsoft: http://www.microsoft.com/casestudies/casestudy.aspx?casestudyid=49127

Microsoft. (2009). *Team Foundation Linking Basics.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/bb130164(VS.80).aspx

Microsoft. (2009). *Team Foundation Team Projects.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms181234(VS.80).aspx

Microsoft. (2009). *Visual Studio Development Environment Model.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/de-de/library/bb165114.aspx

Microsoft. (n.d.). *Visual Studio Team System 2008*. Retrieved January 31, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/vsts2008/products/default.aspx

Minium, D. (2006, January). *Team Foundation Server Fundamentals: A Look at the Capabilities and Architecture.* Retrieved February 1, 2009, from MSDN: Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms364062.aspx

MKS. (n.d.). *Application Lifecycle Management Solutions and more from MKS*. Retrieved January 31, 2009, from MKS: Application Lifecycle Management(ALM), Software Change and Configuration Management (SCM), ITIL Solutions and More: http://www.mks.com/solutions/index.jsp

nexB. (2008). *TeamALM*. Retrieved February 1, 2009, from nexB - The Opne Source ALM Company: http://www.nexb.com/corp/teamalm.html

Oracle Eyes the ALM Market . (2007, August 07). Retrieved February 1, 2009, from CM Crossroads - CM Crossroads the Configuration Management Community: http://www.cmcrossroads.com/content/view/8742/187/

Orcanos. (n.d.). *Application Lifecycle Management, ALM 2.0 Software Development Lifecycle*. Retrieved January 31, 2009, from http://www.orcanos.com

Orcanos. (2008, November). *Proactive Application Lifecycle Management.* Retrieved February 1, 2009, from Application Lifecycle Management, ALM 2.0 Software Development Lifecycle: http://www.orcanos.com/uploaded/brochures/Proactive%20Application%20Lifecycle%20Management.pdf

OSGi Alliance. (n.d.). *About OSGi Technology*. Retrieved July 2, 2009, from http://www.osgi.org/About/Technology

Pasero, B. (2008, October 22). *Using & Extending the Notifier for Events.* Retrieved February 1, 2009, from Jazz Team Wiki: https://jazz.net/wiki/bin/view/Main/FoundationNotifierTutorial

Polarion. (n.d.). *Polarion® ALM — Everthing you need in one single ALM solution.* Retrieved February 1, 2009, from Polarion Software: Application Lifecycle Management, Requirements Management, & Team Collaboration Software Solutions: http://www.polarion.com/products/alm/index.php

Rich, S. (2008, July 7). *The smoothest end game ever… but why?* Retrieved February 1, 2009, from Jazz Community Site: https://jazz.net/blog/index.php/2008/07/07/the-smoothest-end-game-ever-but-why/

Rivieres, J. d. (2007, November 12). *Charter for Jazz REST Services.* Retrieved February 1, 2009, from Jazz Team Wiki: https://jazz.net/wiki/bin/view/Main/JRSCharter

Saad, S. (2008, November 16). *Team Foundation Server: Lessons Learned Through Dogfooding.* Retrieved February 1, 2009, from http://cid-1d5bb72cc739fed7.skydrive.live.com/self.aspx/Public/DVP311|_Saad|_TFSDogfooding.pptx

Serena Sofware. (n.d.). *ALM - Distributed & Mainframe*. Retrieved 01 31, 2009, from Serena Software - Application Development & Business Software Solutions: http://www.serena.com/products/alm/index.html

Teamprize. (2008). Retrieved February 1, 2009, from Teamprize: Team System for Everyone: http://teamprise.com/

TFS Migration and Synchronization Toolkit. (2008, September 16). Retrieved February 1, 2009, from CodePlex - Open Source Project Hosting: http://www.codeplex.com/MigrationSyncToolkit