the WINNOWER

COMPUTER SCIENCES

Towards Abstraction and Automation in Software Engineering

MICHAEL GURSCHLER [1], HENRY EDISON [1], KALLE LAUNIALA [2], PEKKA ABRAHAMSSON [1]

1. Free University of Bozen-Bolzano, Italy

2. ProtonIT, Finland

ABSTRACT

Novel software development approaches are embracing abstraction and automation techniques. It is claimed that abstraction and automation techniques increase the productivity, improve the reusability and lower the complexity of the projects. In this study we address these new frontiers of software development by investigating on one novel proposal, namely the Ball. The Ball is an information ecosystem for authorised information containing web content, digital content as well as service development and integration. It is claimed to improve the reusability, productivity and security of software development while lowering the complexity. While improving the software developer's productivity it should produce smaller and more reasonable software systems, leading to a better reusability and a shorter learning phase for new developers. Up to now there exists no evidence to support these claims. In this study we analyse the Ball ecosystem from multiple perspectives. We compare it to related approaches in order to find its advantages and disadvantages. In order to provide empirical data we replicated a study where a mobile information system was developed using three different technologies. The results of this study show that the Ball ecosystem has the potential to improve the productivity of software development. However, it still needs further development and improvements before being competitive with traditional ways of developing software.

CORRESPONDENCE:
kalle.launiala@protonit.net

DATE RECEIVED:
June 11, 2015

DOI:
10.15200/winn.140836.62974

ARCHIVED:
August 18, 2014

KEYWORDS:
reusability, abstraction, automation, ecosystem, complexity

CITATION:
Michael Gurschler, Henry Edison, Kalle Launiala, Pekka Abrahamsson, Towards Abstraction and Automation in Software Engineering, *The Winnower* 2:e140836.62974 ,

1 INTRODUCTION

The productivity in software development has increased in the last decades. However, the rise of software developers is not sufficient to satisfy the demand for developing new systems or maintaining existing systems [(4)]. The only factor which influences the productivity enough to close this gap is the number of instructions [(4)]. One way to improve the productivity is to search for ways of writing code which are faster, the more effective way is to minimise the number of instructions which are necessary to achieve a certain goal [(4; 24)].

Software systems are becoming bigger and more complex. It often ends up in an extremely complex and clustered system which requires a lot of effort for maintenance. Simple changes or bug fixes require a lot of thinking and deep understanding in the system architecture. NATO Software Engineering Conference [(26)] introduced the concept of software reuse to improve productivity which was the main issue in software crisis in 1968. In this concept, instead of building the system from the scratch, the new system is assembled from the existing and smaller system. Unfortunately, software reuse failed to become a standard software engineering practices.

The main issues with software reuse are related to finding the reusable components, determining the suitability of the components with the needs and using them in current needs [(24)]. In this paper, we examine the Ball, an information ecosystem developed by Kalle Launiala and Jeroen Carelse to tackle this weakness. It is a publishing system for authorised information containing web content, digital

2014 , DOI: 10.15200/winn.140836.62974

content as well as service development and integration. By having a very simple and straightforward structure it is removing complexity and thus it avoids having unnecessary problems. Moreover, it is much simpler to understand how the system works and thus it makes the task of adding new features or changing existing ones much simpler.

Recently, the specific approach of Ball got interest by the industry e.g. Microsoft, Nokia and other companies [22]. Moreover, it looks very promising and there is no similar research of the Ball until now. It has been claimed that the Ball is adding a value to software development. The goal of this paper is to analyse if the Ball ecosystem is able to bring the software development to a new level of reusability, modularity, safety and scalability. By analysing the system from multiple perspectives and comparing it with similar approaches we are trying to find the advantages and disadvantages of this approach. Moreover, we are evaluating it on a real project and comparing it with the traditional way of creating software projects.

The remainder of this paper is structured as follows. In section 2 the main principles of software reuse and software architecture is described. Section 3 discusses the Ball architecture. Section 4 presents the analysis of the Ball. Conclusion and future work are covered in Section 5.

2 STATE-OF-THE-ART

In this section, we describe the state-of-the-art of software reuse and software architecture as the basic foundation of this study.

2.1 SOFTWARE REUSE

There are four main principles in software reuse: abstraction, integration, selection and specialisation [19]. Abstraction is a concept, hiding unnecessary information from the developer and emphasizing important information. Without abstractions the developers would have to filter through different artefacts in order to understand what each of them did and when or how they can be reused efficiently[19]. To join all artefacts, common software reuse technologies usually have an integration framework. The framework can is used to combine the artefacts into a software system, e.g. the module interconnection language [8; 25]. The language enables to export the functions from the producer modules to the consumer modules [19].

However, software reuse has three main issues tthat must be solved in order to be successful[3]. First issue is finding components. Instead of just providing exact matches the system should also provide components with similar features. Those components could then be adapted to the current situation which is more efficient and less error prone then redoing it from scratch [3]. Prieto-Diaz [25] proposed a classification scheme for a component library which should tackle this problem.

Once component with similar features is found, it is required to assess whether a component has to be modified or how the component is used properly. If this is now clear components might be used in a wrong way leading to an unexpected behaviour [3]. Approaches for this issue are hypertext systems like Neptune [7] providing a tool for building a web of information integrating text, diagrams and other information [3]. The dream of reusing software without making modifications is over optimistic. In a reusability system those components are more a living system then a static library. As the system grows and requirements are changed new components will be created or existing components will be updated [3].

The composition of components is very challenging since it must provide a dual character[3]. On one side the composite structures must be represented as independent artefacts with clear characteristics. On the other side there must also be the possibility to further compose them in a new structure with other characteristics. Volpano proposed an approach [33], called Volpano's Software Templates, which divides the information in two parts, namely the data implementation decisions and the essential algorithm. This distinction between the algorithm and the actual implementation for the data type allows a customised implementation of the algorithm [3].

Several approaches, being proposed in the last years, are targeting the issues with the actual software reuse approaches. We will now have a look at different approaches and compare them in order to find the strengths and weaknesses of each of them.

2.1.1 LANGUAGE FOR UNIVERSAL CONNECTOR SUPPORT (UNICON)

UniCon is a component and connection based architectural description language that support several built-in types [28]. Connector defines the protocol for the interaction between components and additional mechanisms e.g. data structures or the initialisation. The protocol ensures the validity of interaction by defining roles and responsibilities for the various components. The roles specify which types of components a connector can interact or handle. Examples for such built-in connectors are the piping mechanism, file access, procedure calls, remote procedure call and many more. The advantage of this approach is that the responsibilities of the components are clearly defined and it is a straight forward process to connect them using the connectors. But the clear definitions for the components and connectors require a lot of effort. Moreover, this approach requires a custom compiler in order to create the systems. This approach was proposed in the mid-nineties but was never adopted widely.

2.1.2 MODEL-DRIVEN ARCHITECTURE (MDA)

MDA consists of guidelines for structuring specifications using Platform Independent Model (PIM) and Platform Specific Model (PSM) [29]. Once the independent model defined, it is transformed into a platform specific model by specifying more details of the underlying system. The first step is to create a platform independent application model represented by the blue circle in the middle of the figure. The platform specialists will then convert the platform independent model into a model which is targeted to a specific platform such as Java, .NET, CORBA, represented by the orange circle. These mappings into a platform specific model can be partially automated by using tools. However, some hand coding is often necessary[29]. As the users experience and the maturity of the tools evolves, the additional effort of hand coding can be reduced.

2.1.3 FRAMEWORK EDITOR (FRED)

FRED approach [11] is a technique to define the specialisation interface of a framework. Using this description, a task-driven application development environment for the framework specialisation can be generated. Basically the definition of the specialisation interface is a specification of a recurring program structure with a precision e.g. a recipe. It is an abstract structural description of an extension point of a framework. It is given

2.2 SOFTWARE ARCHITECTURE

Software architecture is a set of structures dealing with the design and implementation of the high-level structure of a software system. It must satisfy the requirements of the systems ranging from functionality to non-functional requirements e.g. scalability, reliability, portability or availability [18]. Another definition of by Garlan and Shaw [9], software architecture is the overall system focusing on the high level abstraction which consists of components and connectors. Jansen and Bosch instead call the software architecture a set of architectural design decisions that lead to a set of structures [16].

2.2.1 4+1 ARCHITECTURAL VIEW MODEL

The 4+1 view model was proposed by [18] in 1995. This model describes the architecture of software systems using multiple views: logical view, process view, physical view, development view and scenario. These views address the functional and non functional requirements of different stakeholders e.g. the end-user, developers, system engineers, project managers [18]. Krutchen proposes multiple views or perspectives in order to address large architectures. This model was used with success on different projects with no or only a few customisations and adjustments [18].

2.2.2 SEPARATION OF CONCERNS

ARES System of Concepts (ASC) [2] is a a conceptual framework developed by Nokia. It is dealing with the separation of concerns in order to manage the complexity of the architecture design. ASC is based on the separability of different segments in the software transformation cycle (design, build, upgrade and runtime)[13]. ASC focuses on the design of texture addressing the concerns which cannot

be detected with the main structure. In this approach the software architect analyses the design decisions such as the platform, road maps and requirements and produces architecturally significant requirements. Moreover, it groups those architecturally significant requirements by the segments of the software transformation cycle that they address[13].

2.2.3 SIEMENS' 4 VIEWS (S4C)

S4V approach was proposed by Siemens Corporate Research[30]. It is based on the best practices for industrial systems. It consists of four different views: conceptual view, execution view, module interconnection view and code architecture. The views of S4V are developed using the Global Analysis activity where the architect specifies the technological, organisational and product factors influencing the software architecture (system qualities, organisational constraints, requirements, existing technology)[13]. Using these factors the architectural issues can be identified. The issues often arise when a set of factors will be difficult to fulfil together. Later on design decisions are applied on one or more views in order to solve the issue.

2.2.4 ENTERPRISE ARCHITECTURE (EA)

EA is defined as a "fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" [15]. An enterprise is a concept, not a physical reality. It can be seen as a strategic entity which is built of physical components but is itself not a physical entity [12]. EA is often assigned to an enterprise architect who is responsible for the development and maintenance [12]. EA approaches are becoming more important since they are capable of aligning the IT and the business in organisations [5]. This alignment of the IT and the business improves the communication between the IT staff and the executive stakeholders and allows business visions to be represented in operations and systems [1; 32]. Moreover, they bring a benefit in the alignment between strategy and execution, operations and infrastructure, and collaboration among planning[10; 20]. A drawback of this high-level nature of EA approaches is that it requires experienced personnel on EA side and on the application domain since the adoption to a specific domain is a very challenging task [1]. Moreover, EA approaches often propose models which do not express the organizational goals or context, requiring a tailor-made specific solution for the current situation which leads to a limited interoperability with other components of the architecture [1].

2.2.5 THE OPEN GROUP ARCHITECTURE FRAMEWORK (TOGAF)

TOGAF is a high level EA framework which was proposed by the Open Group [31]. It states the possibility to manage the creation and management of an architecture and the architectural descriptions themselves. It focuses on critical business applications which are using building blocks of the open systems. TOGAF divides the architecture in the following categories: business architecture, application architecture, data architecture and technical architecture [31; 27].

2.2.6 ZACHMAN FRAMEWORK

Zachman framework was published by John Zachman, one of the pioneers in the enterprise architecture domain [23]. According to Zachman [35], the increasing level of complexity and scope of design in information systems make the use of an architecture necessary. Zachman Framework relies on principles of traditional architecture establishing a set of views describing a complex enterprise system. Zachman framework does not provide a guidance for the sequence, implementation or process, but it ensures that all perspectives are well established. The outcome ends in a complete system, regardless in which order the views where implemented [27]. The framework doesn't help much in deciding if the outcome is the best possible architecture [27]. It is basically the result of applying a methodology to a specific enterprise.

2.2.7 FEDERAL ENTERPRISE ARCHITECTURE (FEA)

FEA approach is an approach trying to unite the agencies and functions of the government using a single enterprise architecture [6]. FEA unites the features of Zachman framework and TOGAF e.g. the comprehensive taxonomy and the architectural process [27]. FEA perspective on the enterprise architecture consists of segments being a business functionality like for example the human resource [27]. All the architectural segments are implemented individually by following structured guidelines.

Each of the segments is considered to be its own enterprise within the federal enterprise. The FEA allows a flexible usage of the methods, tools and work products in the federal agencies.

2.3 ABSTRACTION DESIGN METHODOLOGY (ADM)

ADM is a methodology that allows abstraction of any information [22; 21]. The core of this approach is to built around the rapid creation of reusable abstractions (design decisions) by system designers. This approach comprises three different elements:

- •
  Definition of templates which will be used as moulds for abstraction and describe how properties of abstractions are related and what items are needed for the creation of a new abstraction.
- •
  Definition of transformations or generators that transform the templates into a runnable code.
- •
  Description for abstraction instantiation that describes how the abstractions can be instantiated and the configuration that is being created.

These different elements require individual competences and skills. Therefore ADM may imply a redefinition of the software roles. Each element can be implemented by a specialist in a specific area. Software architects might focus on the template development whilst software specialists take care of the code generation and application engineers are responsible for the actual application design.

ADM unifies the advantages of different reuse practices. It maintains the flexibility and modularity of libraries but removes the risk of having unintended behaviour since there is no hidden code. A practice, having several disadvantages due to a bad traceability and maintainability, which is close to this approach is copy-paste coding. Another disadvantage of copy-paste coding is that it often introduces an unexpected behaviour. The use of wizards allows a fast code generation but it is difficult for the developer to understand what is happening until the code is actually generated. Moreover, it results often in a complex system which is difficult to figure out what is happening at which point of generation.

ADM is trying to combine these two principles in an innovative way. It is similar to both of the approaches. It makes use of a piece of code which is used as a template for generating other code. These templates are formalised using a different formalism which can be adapted to a particular situation. ADM is also making use of wizards which are in a form of scripts and XML definitions. Those scripts are used for the automatic code generation. The developer has the possibility to adapt the scripts or definitions in order to fit its needs.

ADM provides a way to modularise software architecture at source code level. In terms of introducing another view, it removes unnecessary abstract view to development. It enables to automate the platform level coding based on the higher level abstraction. Compare to other approach, ADM does not require additional tool. It has one XML-scheme to specify the language and the content.

3 BALL ARCHITECTURE

The Ball is an information ecosystem developed by Kalle Launiala and Jeroen Carelse. It is a publishing system for authorised information containing web content, digital content as well as service development and integration. It is originated from information models which do not constrain what to put into the system and how to process it. This context insensitivity makes the system very flexible and usable for many different scenarios. It consists of the following elements:

- •
  Information is stored and recalled from its internal storage being a database, a blob storage or another medium.
- •
  An atom is the smallest unit of executable code e.g. a button or a combo box.
- •
  A module is a combination of multiple atoms to achieve a specific task.
- •

An instance is a combination of multiple modules e.g. a mobile or a web application.

- •

An interface is the medium through which the user experiences the instance e.g. a smartphone, a tablet pc or a notebook. The same instance may be visualised differently and optimised for specific device. When the user interacts with the interface the information will go through the instance up to the database and the processed information then comes back in various feedbacks e.g. text messages, on-screen information.

All elements can be shared within the same, or across different Balls. Moreover, it is possible to remove or add new elements without recreating everything from scratch, e.g. a new module to an instance. LABEL: shows the structure of a Ball.

A Ball can also be cloned and stored in different places (see LABEL:). In real world, an application or a service can be seen as a small instance of the Ball which can be cross connected with other instances of the Ball. Having this well-defined but simple interface for the cross connection it is possible to share information in a way such that the user will not even notice if the information comes from another Ball.

FIGURE 1: OVERVIEW OF THE BALL

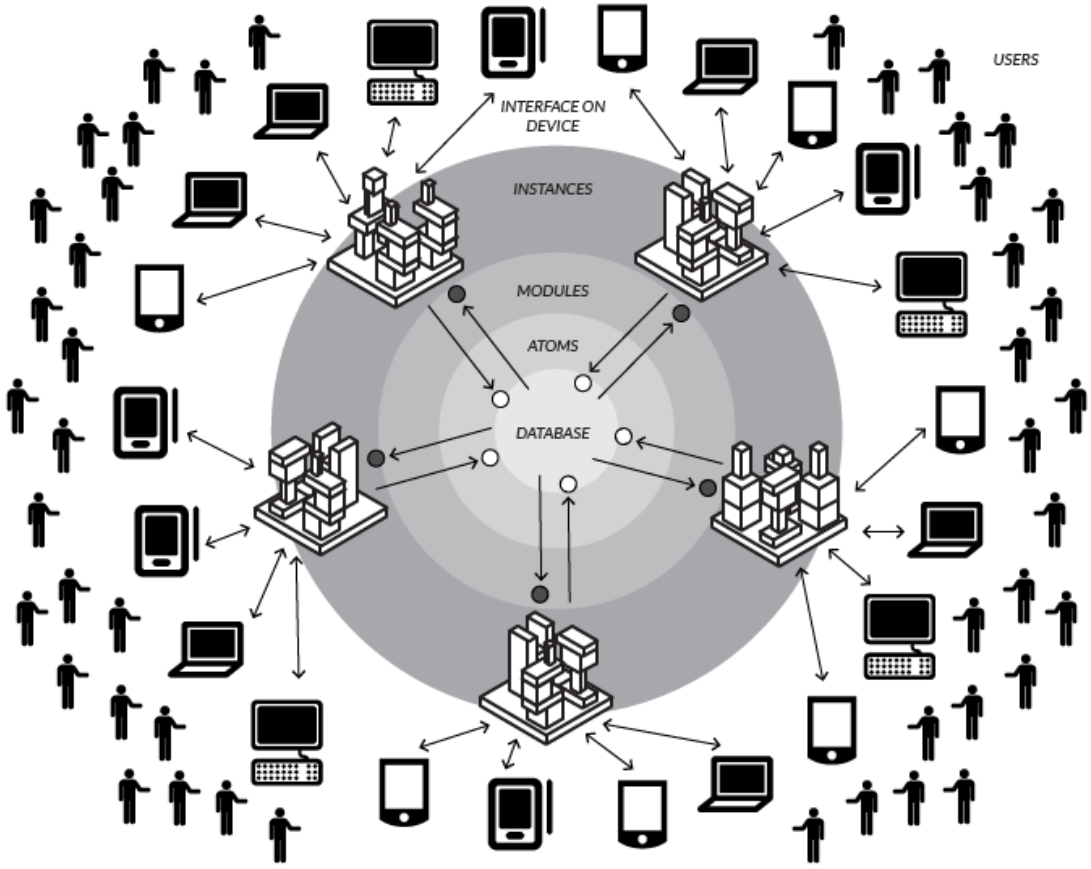
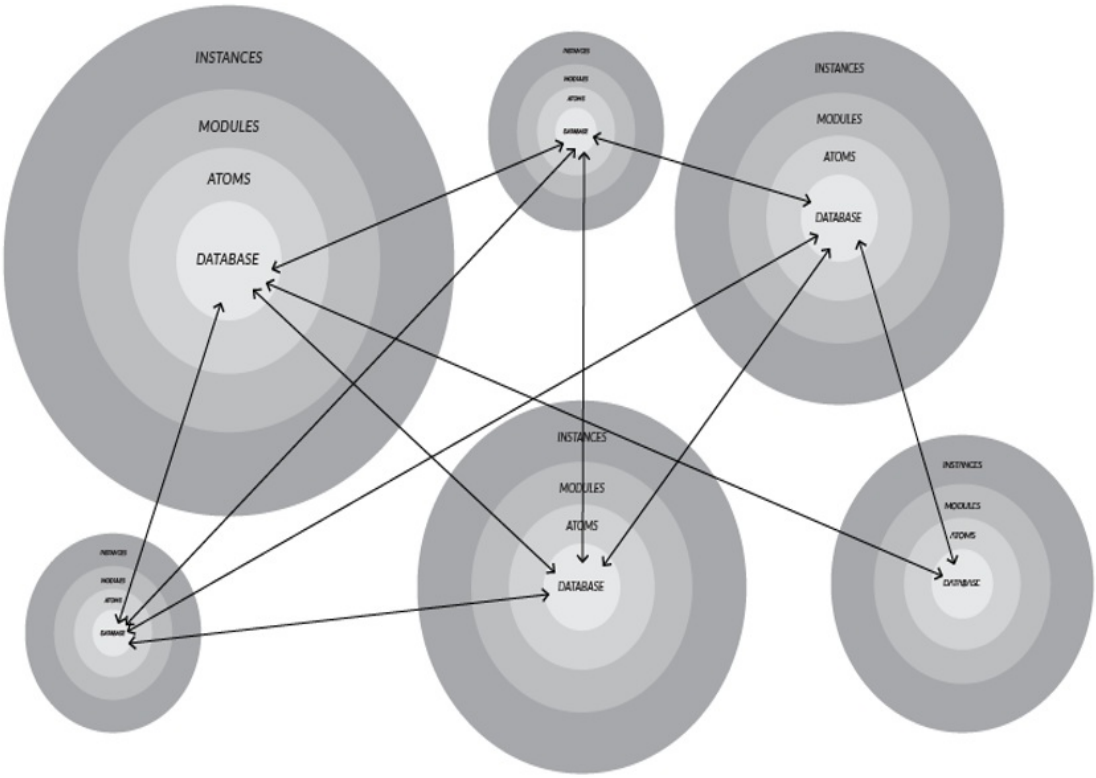


FIGURE 2: INTERCONNECTION OF BALLS



3.1 LOGICAL ARCHITECTURE

The core system of Ball is System Operating System (SOS). It can be seen as a traditional operating system without handling the access to the hardware layer. SOS provides features like security boundaries, memory management, computational power management and authorisation management. SOS is context sensitive but content insensitive. It does not constrain the content and information to be processed by the system. However, it enforces some design constructs to be there. Thus, systems which build on top of SOS can rely on them. This does not limit the productivity of the developers since the similarities in each of the systems makes it much easier for the developers to understand how the system behaves. The building blocks of the Ball are as follows:

- •

  The information model which reminds the class models with namespaces. LABEL: shows a streamlined example for an information object encoded in the XML structure representing the MyCustomer type. It consists of an id and the name.

  LISTING 1: STREAMLINED EXAMPLE OF THE INFORMATION MODEL

```
<InformationObject name="MyCustomer">
 <InformationItems>
 <InformationItem name="Id" logicalDataType="Number_Integer" />
  <InformationItem name="Name" logicalDataType="Text_Short" />
 </InformationItems>

</InformationObject>
```

- •

  The process model which is a service method model with orchestration sequence. LABEL: shows a streamlined example for a process model encoded in the XML structure representing the AddMyCustomer operation which first creates a new object of the type MyCustomer which was defined in the information model. Then it sets the id and the name of the object to the given values and writes the object to the storage.

  LISTING 2: STREAMLINED EXAMPLE OF THE PROCESS MODEL

```
<Operation name="AddMyCustomer">
```

```
<Parameters>
 <Parameter name="Id" dataType="Number_Integer" />
 <Parameter name="Name" dataType="Test_Short" />
</Parameters>
<Execution>
 <SequentialExecution>
  <TargetDefinition dataType="MyCustomer" name="CreatedObject" />
  <MethodExecute name="SetCustomerId">
   <Parameter name="Id"/>
   <Parameter name="Name" />
   <Target name="CreatedObject"/>
  </MethodExecute>
  <MethodExecute name="StoreObject">
   <Target name="CreatedObject"/>
  </MethodExecute>
 </SequentialExecution>
</Execution>
</Operation>
```

- •

  The ADM modules which concretise information and process models to a service interface (a platform agnostic, native code), a class model (serialised to storage) and a native code execution sequence.

The Ball provides the necessary functionalities to discover services as in the real world. In order to discover the right service the Ball is doing a so called semantic interface mapping. Consider two applications that have Customer class which are semantically similar but not equal. Therefore, it is necessary to use the technical information of a service method, including the return value, the method name and the parameters and to introduce an additional information layer. Hence it is possible to calculate a unique hash value. This value will be matched when searching for a service with the same hash value. In addition to this there are also tiny service which map the information between two similar types. Thus it is possible to use a service even if the signature is not a perfect map (e.g. convert MyApp.Customer to YourApp.Customer).

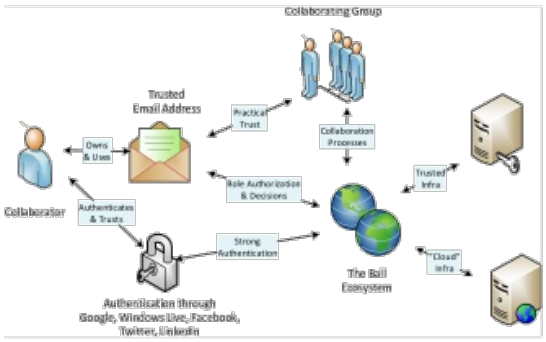
The Ball architecture can be implemented using existing techniques. The traditional class and domain model becomes the information model and the process models, activity diagrams or process flow models become the operation model. It is focusing on small contexts and the complexity of the system is kept at acceptable and relatively small level. The large systems are a combination of small subsystems with explicit, complete and up to date declarations of the information and operation model. It is capable of handling the complexity, arising in large software systems, at the internal of the Ball. The biggest advantage of this approach is no need for additional tooling and methods to handle the large views. Everything is declared in a complete and automatically up to date fashion using the information and operation model. It is also capable of generating automatic, dynamic search results of existing facts which can be categorised in order to become dynamic models.

3.2 SECURITY AND PRIVACY

Data security and privacy is a crucial part in many nowadays software systems. Often it is very challenging to guarantee that data can only be accessed by authorised users. Moreover, the data has to be protected from malicious corruption. The Ball provides a simple but powerful solution for these problems. LABEL: shows that the authentication and authorisation of a collaborator are completely separated. The technical authentication is delegated to experts using an open id provider like Google, Yahoo, Windows Live, etc. or even a stronger provider for the bank authentication level. The authorisation is done using the trust token, e.g. as the email address, of the collaborator. This email address is validated using a confirmation mail before it is used as the trust token. If an email gets invalidated for some reason, all the authorisations and memberships of groups are invalidated as well

until the email address is again validated. Collaborators can be invited to a group by adding their email address. Once the invitation was accepted the collaborator has access to the group's information.

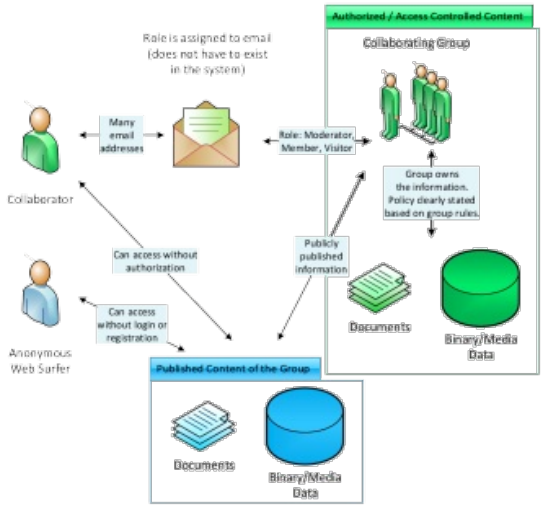
FIGURE 3: COLLABORATION IN THE BALL



As shown in LABEL: collaboration groups owns the information of any type e.g. documents, binary data, or media data. All the information has the same security context and is protected from unauthorised access. All processing of the information is done within this security context and no leaks outside. To get authorised access within a group they have to be invited by one of the existing members. Each collaboration group has also a completely public area to publish the information. This information can be retrieved by other users, regardless if they are a collaborator of the group or if they are an anonymous user.

In SOS the user is a role rather than an individual. Therefore, they have certain permissions to interact with other based on their roles. SOS enables these constraints to be digitally accelerated while still offering a transparent fall-back to the non-digital world. In SOS information has always an owner. This owner must not be the real copyright holder which legally owns this information. The owner is the person who received this information. If a person receives a confidential presentation it has an authorisation to further use or redistribute it, even though it is not the legal copyright holder.

FIGURE 4: AUTHORIZATION CONTEXT IN THE BALL



3.3 KNOWLEDGE GAPS

The Ball is targeting the same problems as the approaches described in LABEL:. Instead of introducing complex mechanisms and structures to solve certain problems, it combines existing constructs and removing unnecessary ones to reduce complexity. Thus, it eliminates the problems before they occur. While other software architecture approaches are trying to have all information in their control they could become extremely complex. This might lead to scenarios where the software developers might lose the focus from the actual problem. Moreover, this high complexity leads to new problems which

would not exist if the complexity is kept at a reasonable level.

Because of the very simple and modular structure it is also targeting the reusability and abstraction problems. Often different abstraction or reusability approaches do not work together nicely which either leads to additional effort or has an impact on the advantages of one of the approaches. The Ball instead is capable of combining the constructs of different approaches in order to unite the advantages without having to many disadvantages.

3.4 CURRENT STATUS AND KNOWN LIMITATIONS

The Ball is new and innovative. Hence, it requires a certain understanding to see its benefits before the developers are able to use it. Once the software developers get the grasp of the concept they will understand how the system behaves. Even though the Ball is freely usable for everybody this issue might slow its spreading down. Since it is in heavy development, there are many things which do not work as intuitive as they could. Tasks like running the code generation or uploading templates to the Ball require to start an external console application by hand.

Currently, the Ball is bound to Windows Azure Cloud. Hence it is not possible to run code, like Java or C++ which are not natively supported by the Azure cloud. Porting the Ball to another platform like the Amazon Cloud is a straight forward task but up to now it is only supporting Windows Azure. Moreover, cloning the existing Ball to a new Windows Azure account requires additional work. Due to the lack of a documentation and unforeseeable incidences it might be a difficult and time consuming task to hook it up.

Information in the Ball is stored redundantly. Having different items which belong to a MasterCollection results in one file for the collection which contains all the items (with their whole content) and one file for each item which contains its content. Thus, the same information is stored in multiple places. The same happens if information is shared between multiple accounts. This leads to the fact that much more blob storage space is required then expected.

4 EXPERIENCE WITH THE BALL ECOSYSTEM

In this chapter we describe the experience working with Ball ecosystem.

4.1 RESEARCH DESIGN

In our study we used a comparative case study as described in [34]. This method is used to check the variations in different environments. Hence, it can be used to compare objects which are similar in some aspects but differ in other aspects. In order to have a meaningful comparison it is necessary that the objects have some aspects where they are similar. Since we have a scenario which matches these constraints we decided to use a comparative case study in order to compare the evaluation projects.

For this study, a stock exchange application is developed[17] using three different technologies: zOmbie, Naked Zombie and Titan. These applications are using a server to retrieve the stock market information from public sources and storing it into a data store in regular intervals. The user has the possibility to access the data using a web page and can add personal information about his/her current investments. The system will then provide their current value and the daily changes. Moreover, the user must be able to set alerts for stock prices or trade directly in the system. All three projects share different commonalities because they have the same features but differ in other points like the user stories, the time frame, the technical environment or simply the number of people which are involved in the project (see LABEL:). In order to evaluate the development of our projects the actions during development have been tracked with a five minute precision. The data was registered using an excel sheet and updated every day in the repository. As proposed in [14], we collected the following information :

- •
  Time: the time which was spent for the work.
- •
  Size: the size of the part which was done.

- •
  Defect: when and where defects where injected, found and fixed.
- •
  Date: the date when the work is done.
- •
  User story and task: the user story and task to which the actual work belongs.
- •
  Category: the category to which the work belongs (Planning and Management, Coding, Bug fixing and Quality Insurance or Testing)

TABLE 1: TECHNICAL ENVIRONMENT

Item	zOmbie	Naked Zombie	Titan
Server side language	Java 2 SDK 1.4.1	Java 2 SDK 1.4.2	C# .NET 4.5
Client side language	Java MIDP 1.0	Java MIDP 2.0	HTML JavaScript
Web server	Tomcat 4.1	-	Windows Azure (IIS)
Framework	-	Naked Objects Framework 1.2.2	The Ball
Database	MySQL 4.0.9 + Java connector	XML Object Store	Windows Azure Blob Storage
Development Environment	Eclipse 2.1, Nokia Dev. Suite 2.0 for J2ME	Eclipse 3.0.1, Eclipse ME J2ME Plugin	Visual Studio 2012, JetBrains WebStorm 6.0.2
Software Configuration Management	CVS (1.11.2); integrated to Eclipse	GitHub, Git Extensions	
Documents	MS Office XP	MS Office XP, Rational Rose	MS Office 2013
Platform	Mobile Application	Web Application	

4.2 RESULTS AND DISCUSSION

LABEL: shows that the productivity in the Naked Zombie project was better than in the zOmbie project. Due to the fact that only 57% of the features have been developed in the Titan project we had to estimate the relative effort which would have been required to implement all features.

TABLE 2: COMPARISON OF THE PROJECT DATA

Collected data	zOmbie	Naked Zombie	Titan
Time in weeks	8	4	2
Number of involved persons	4	4	1
Total effort in hours	1073	383	102
Percentage of implemented features	100%	100%	57%
Relative effort	1073	383	165 to 195[11]The relative effort is estimated with the current productivity and the

Collected data	zOmbie	Naked Zombie	Titan standard error calculated in LABEL:
Effort comparison	0%	-64%	-82% to -86%

4.2.1 EFFORT DISTRIBUTION

In terms if effort distribution, both zOmbie and Naked Zombie projects have a similar trend. Only the amount of time which was used for the bug fixing was remarkably larger in the Naked Zombie project then in the zOmbie project due to problems with the Naked Objects framework. Titan project has a different effort distribution. Less effort is used for the planning and management since the Ball framework is already providing the system architecture by generating most of the code automatically. Thus, only small code fragments have to be implemented which does not require as much planning upfront. Also the coding in the Titan project required less effort since a lot of intermediate work which was done in the other project was generated due to the power of ADM code generation. Moreover, it enforces certain design constructs, resulting in a straightforward system which contains certain functionalities out of the box.

Unfortunately, the effort for bug fixing and quality insurance was about 38% of the total effort which is much more than in the zOmbie (9%) and in the Naked Zombie (20%) projects. Due to the architecture of the Ball and the fact that it is still under heavy development the developer faced many different problems and bugs. Most of them are either caused by the incomplete Ball environment or the wrong usage of it.

4.2.2 USER STORY EFFORT

Even if the projects have the same functionality, they have different user stories. Thus, it is not possible to directly compare them. Nevertheless the user story effort, the number of user stories and the speed of implementation from the customer point of view can be compared as shown in LABEL:.

TABLE 3: USER STORY EFFORT

	zOmbie	Naked Zombie	Titan
Number of user stories	22	12	6
User story effort (median)	18,9	9,7	10,5
User story effort (max)	40	33,2	21
Implemented features	100%	100%	57%

Since the functionality in the zOmbie and Naked Zombie project are approximately the same, we see in LABEL: that the user stories of the Naked Zombie project provided more value to the customer. Moreover, the implementation of the user stories took less time than in the zOmbie project. In the Titan project only 57% of the functionality have been implemented. The median effort for the user stories is slightly bigger than the median effort in the Naked Objects project. But since the user stories of the Titan project provide more value to the customer it leads to a better development speed.

TABLE 4: ESTIMATION ACCURACY

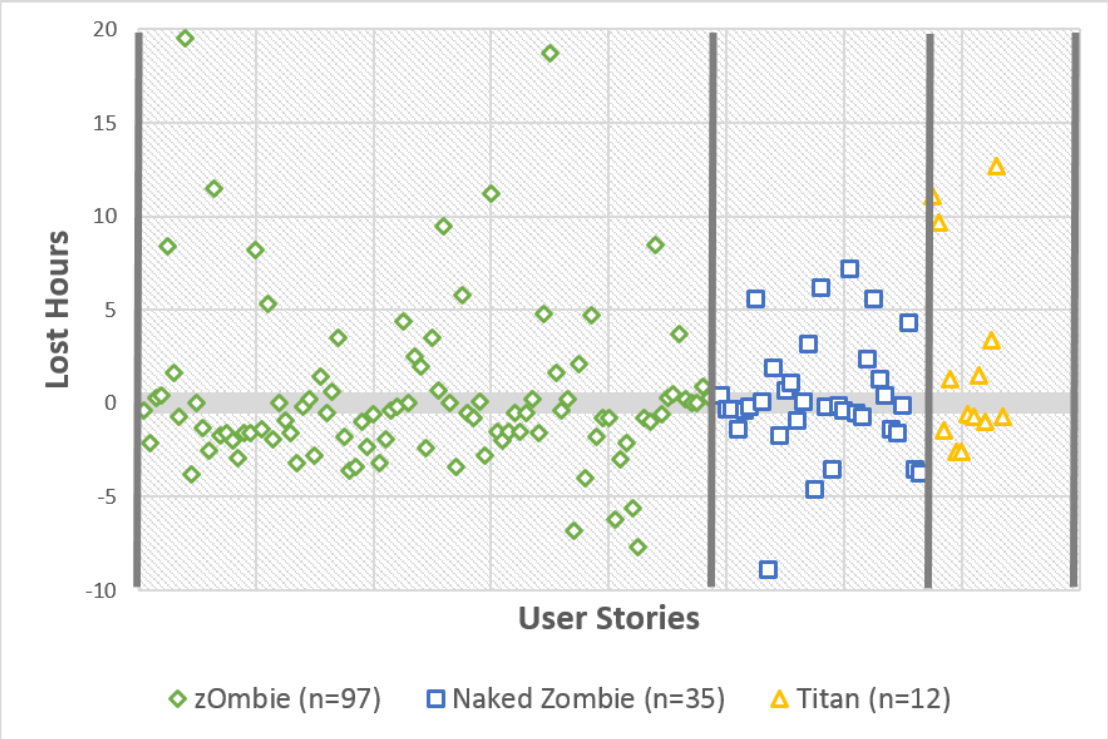
Estimation Accuracy	zOmbie	Naked Zombie	Titan
Median	-19%	-31%	+7%
Minimal	-100%	-82%	-65%
Maximal	+400%	+435%	+318%
Standard Deviation	28%	35%	122%

4.2.3 EFFORT ESTIMATION ACCURACY AND PRECISION

The estimation accuracy is shown in LABEL:. The negative values indicate an overestimation and the

positive values indicate an underestimation of the tasks. From the table, we can see that the estimation accuracy of the zOmbie and Naked Zombie project are at a similar level. Both projects tend to overestimate the tasks slightly and also the standard deviations are quite low and therefore the absolute error range measured in actual hours is acceptable. In Titan project, instead the tasks have been slightly underestimated. It means that on average they took longer than expected. The standard deviation of the estimation accuracy in the Titan project is much higher than in the other projects. The fact that so much bug fixing was needed during the development might be the reason for this big error rate.

FIGURE 5: HOURS LOST BY WRONG ESTIMATES IN THE PROJECTS



By analysing LABEL: we see that the development of the zOmbie and Naked Zombie project lead to a similar result. Most points are located around the zero loss line. Only few tasks had an implementation effort which differs significantly from the estimation. The Titan project instead has only few points which are located around the zero loss line. Most of the tasks either took much more time or much less time than expected. This big variance of the values is explained once again due to the upcoming problems which had to be fixed in order to complete the task.

We realised that due to the limited availability of documentation, it was necessary to get additional help by the inventor of the Ball. Although, ADM has made it simpler by removing unnecessary views, the learning curve by unexperienced programmer in ADM gets higher. The Ball is not yet mature and thus the developer has problems to understand how it actually works and how to use it properly. This leads to the difficulty to estimate the effort needed to complete a certain task. This relatively big estimation error is resulting in a big cost estimation error. Thus, a company might overestimate or underestimate the costs. This problem limits the company in their flexibility because of the un-precisely plan.

5 CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In this study novel abstraction and automation approaches in software development have been compared and analysed. During the development phase information was collected and thus it was possible to compare the Ball with the traditional software development. The Ball has a significantly better productivity and the structure of the system is clean due to the fact that the code generation is enforcing simple design constructs.

The Ball is a very interesting system which unites great concepts. But due to the fact that it is not stable enough and is still lacking many features it is difficult to apply the Ball in a real world scenario. But its concepts look promising. As soon as the actual implementation of the concepts is done in a neat and clean way it has the potential to bring the software development to a new level.

5.2 FUTURE WORK

In this section we provide recommendations which will be useful for the future development. First of all the Ball needs a clear roadmap and change log to specify the current status of each features. This will give opportunity for open source community to see and focus on solving different aspect of the Ball. To achieve this, a proper documentation explaining the system is needed. The documentation should also covers the theoretical knowledge or basic concepts and ideas behind the Ball. Another recommendation is that the code should be cleaned and commented where necessary. This will help developers to comprehend the code.

ACKNOWLEDGEMENT

The authors would like to express sincere thanks to Xiaofeng Wang for the advice and suggestion and Daniel Graziotin for making great effort of conducting the workshop at Free University of Bozen-Bolzano.

REFERENCES

- G. Antunes, J. Barateiro, C. Becker, J. Borbinha and R. Vieira (2011) Modeling contextual concerns in enterprise architecture. pp. 3–10. External Links: Document Cited by: 2.2.4.
- L. Bass, P. Clements, R. Kazman and M. Klein (2008) Evaluating the software architecture competence of organizations. pp. 249–252. External Links: Document Cited by: 2.2.2.
- T. J. Biggerstaff and A. J. Perlis (Eds.) (1989) Software reusability: vol. 1, concepts and models. ACM, New York, NY, USA. External Links: ISBN 0-201-08017-6 Cited by: 2.1, 2.1, 2.1.
- B.W. Boehm (1987) Improving software productivity. Computer 20 (9), pp. 43–57. External Links: Document, ISSN 0018-9162 Cited by: 1.
- I. Brown and I. Motjolopane (2005) Strategic business-it alignment, and factors of influence : a case study in a public tertiary education institution. South African Computer Journal 35, pp. 20–28. Cited by: 2.2.4.
- CIO Council (1999) FEAF - Federal Enterprise Architecture Framework Version 1.1. Cited by: 2.2.7.
- N. M. Delisle and M. D. Schwartz (1986) Neptune: a hypertext system for cad applications. pp. 132–143. Cited by: 2.1.
- F. DeRemer and H. H. Kron (1976) Programming-in-the-large versus programming-in-the-small. IEEE Trans. Software Eng. 2 (2), pp. 80–86. Cited by: 2.1.
- D. Garlan and M. Shaw (1993) An introduction to software architecture. Singapore, pp. 1–39. Cited by: 2.2.
- A. Goel, H. Schmidt and D. Gilbert (2009) Towards formalizing virtual enterprise architecture. pp. 238–242. External Links: Document Cited by: 2.2.4.
- M. Hakala, J. Hautamäki, K. Koskimies, J. Paakki, A. Viljamaa and J. Viljamaa (2001) Generating application development environments for java frameworks. pp. 163–176. Cited by: 2.1.3.
- K. Harmon (2005) The ”systems” nature of enterprise architecture. Vol. 1, pp. 78–85 Vol. 1. External Links: Document Cited by: 2.2.4.
- C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran and P. America (2005) Generalizing a model of software architecture design from five industrial approaches. pp. 77–88. External Links: Document Cited by: 2.2.2, 2.2.3.
- W. S. Humphrey (1995) A discipline for software engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. External Links: ISBN 0201546108 Cited by: 4.1.
- [15] (2007) ISO/iec standard for systems and software engineering - recommended practice for architectural description of software-intensive systems. ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15, pp. c1–24. External Links: Document Cited by: 2.2.4.
- A. Jansen and J. Bosch (2005) Software architecture as a set of architectural design decisions.

pp. 109–120. Cited by: 2.2.

- H. Keränen and P. Abrahamsson (2006) Naked objects versus traditional mobile platform development: a comparative case study.. pp. 274–283. External Links: ISBN 0-7695-2431-1 Cited by: 4.1.
- P. Kruchten (1995) The 4+1 view model of architecture. IEEE Softw. 12 (6), pp. 42–50. External Links: Document, ISSN 0740-7459 Cited by: 2.2.1, 2.2.
- C. W. Krueger (1992) Software reuse. ACM Comput. Surv. 24 (2), pp. 131–183. Cited by: 2.1.
- M. M. Lankhorst (2013) Enterprise architecture at work - modelling, communication and analysis (3. ed.). The Enterprise Engineering Series, Springer. External Links: ISBN 978-3-642-29650-5 Cited by: 2.2.4.
- K. Launiala (2012) Microsoft TechDays 2012 Finland ADM Materials. Note: \urlhttp://abstractiondev.wordpress.com/2012/03/09/microsoft-techdays-2012-finland-adm-materials/[Online; accessed 20.09.2013] Cited by: 2.3.
- K. Launiala (2013) “The Ball” Launched @ Microsoft Finnish TechDays!. Note: \urlhttp://abstractiondev.wordpress.com/2013/03/13/the-ball-launched-microsoft-finnish-techdays/[Online; accessed 20.09.2013] Cited by: 1, 2.3.
- F. G. leerstoel (2005) An overview of enterprise architecture framework deliverables. Technical report K.U.Leuven, K.U.Leuven. Cited by: 2.2.6.
- H. Mili, F. Mili and A. Mili (1995) Reusing software: issues and research directions. Software Engineering, IEEE Transactions on 21 (6), pp. 528–562. External Links: Document, ISSN 0098-5589 Cited by: 1, 1.
- R. Prieto-Díaz and J. M. Neighbors (1986) Module interconnection languages. Journal of Systems and Software 6 (4), pp. 307–334. Cited by: 2.1, 2.1.
- B. Randell (1979) Software engineering: as it was in 1968. pp. 1–10. Cited by: 1.
- R. Sessions (2007) Comparison of the Top Four Enterprise Architecture Methodologies. Technical report ObjectWatch. Cited by: 2.2.5, 2.2.6, 2.2.7.
- M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young and G. Zelesnik (1995) Abstractions for software architecture and tools to support them. IEEE Trans. Software Eng. 21 (4), pp. 314–335. Cited by: 2.1.1.
- R. M. Soley (2003) Model driven architecture: the evolution of object-oriented systems?. pp. 2. Cited by: 2.1.2.
- D. Soni, R.L. Nord and C. Hofmeister (1995) Software architecture in industrial applications. pp. 196–196. External Links: ISSN 0270-5257 Cited by: 2.2.3.
- The Open Group (2009) TOGAF 9 - The Open Group Architecture Framework Version 9. The Open Group, The Open Group, USA. Cited by: 2.2.5.
- W. Van Grembergen (2004) Strategies for information technology governance. Igi Global. Cited by: 2.2.4.
- D. M. Volpano and R. B. Kieburtz (1985) Software templates. pp. 55–61. Cited by: 2.1.
- R. K. Yin (1984) Case study research: design and methods. Applied social research methods series, Sage Publications, Beverly Hills, CA. Cited by: 4.1.
- J. A. Zachman (1999) A framework for information systems architecture.. IBM Systems Journal 38 (2/3), pp. 454–470. Cited by: 2.2.6.