ISSN 1330-3651
UDC/UDK 004.415

A COMPARATIVE STUDY OF AGILE, COMPONENT-BASED, ASPECT-ORIENTED AND MASHUP SOFTWARE DEVELOPMENT METHODS

Ahmed Patel, Ali Seyfi, Mona Taghavi, Christopher Wills, Liu Na, Rodziah Latih, Sanjay Misra

Subject review

This paper compares Agile Methods, Component-Based Software Engineering (CBSE), Aspect-Oriented Software Development (AOSD) and Mashups as the four most advanced software development methods. These different approaches depend almost totally on their application domain but their usability can be equally applied across domains. The purpose of this comparative analysis is to give a succinct and clear review of these four methodologies. Their definitions, characteristics, advantages and disadvantages are considered and a conceptual mind-map is generated that sets out a foundation to assist in the formulation and design of a possible new integrated software development approach. This includes supportive techniques to benefit from the examined methods' potential advantages for cross-fertilization. It is a basis upon which new thinking may be initiated and further research stimulated in the software engineering subject field.

Keywords: *agile, aspect, block-based programming, component, mashup, software development, end-user development, Web 2.0, Web 3.0*

Poredbena studija metoda razvoja softvera – prilagodljivih, utemeljenih na komponentama, usmjerenih na gledište i mješovitih (mashup)

Pregledni članak

U ovom se članku uspoređuju metode razvoja softvera – prilagodljivih, utemeljenih na komponentama, usmjerenih na gledište i mješovitih (odnosno Agile Methods, Component-Based Software Engineering - CBSE, Aspect-Oriented Software Development - AOSD i Mashups), kao četiri najnaprednije metode za razvoj softvera. Ovi različiti pristupi gotovo u potpunosti ovise o području njihove primjene, ali im je upotrebljivost jednaka u svim područjima. Cilj je ove usporedne analize dati sažet i jasan pregled ove četiri metodologije. Razmatraju se njihove definicije, karakteristike, prednosti i nedostaci te se generira konceptualna mapa namjera koja daje osnovu za pomoć u formulaciji i razvoju mogućeg novog integriranog pristupa za razvoj softvera. To uključuje tehnike podrške kako bi se moguće prednosti ispitivanih metoda iskoristile za uzajamno poboljšanje. To je osnova na kojoj se može razviti novi način razmišljanja i potaknuti daljnje istraživanje u području softverskog inženjeringa.

Ključne riječi: *agilan, aspekt, programiranje utemeljeno na bloku, komponenta, mashup, razvoj softvera, Web 2.0, Web 3.0*

1 Introduction

The most constructive approach that can be taken to define new methodologies in a domain is that of a comparison between the different available methods in that domain. This valuable comparison will help create new avenues of exploration, which could lead to determining new opportunities.

Since the early 80's, computer software scientists and technicians have been under the impression that the software production process consists of a set of organized and well-defined distinct steps which resembled Management Information Systems (MIS) production processes which have now evolved to different forms of advanced computing services [1]. However, the established approaches to software development are always the subject of continuous review. It is in this spirit of scientific review that we examine four of the latest software development methodologies: Agile methods, Component-Based Software Engineering (CBSE), Aspects-Oriented Software Development (AOSD), and Mashups. We compare them with a view to determine what kind of an approach might be most suitable in the future. Several difficult research questions beyond the scope of this paper have arisen that are discussed in outline but could be pursued in more detailed research work. We attempt to provide some analysis of these research opportunities.

This paper reviews and compares Agile methods, Component-Based SE, Aspect-Oriented SD, and Mashups as the latest four software development methods. Unlike some other methods, these four depend almost totally on the application domain as opposed to different forms of computing. These approaches are also the top-four, according to many criteria such as usage, integrity, tools accessibility, etc. That is the reason why they are considered as pillars upon which the search for a new close-optimal approach is based. The comparison includes a description of the approaches, their characteristics, advantages and disadvantages. We hope that this will help enable the generation of a conceptual mind-map and provide a useful foundation for formulating and designing new frameworks and integrated software development approaches with new supportive techniques as we progress from Web 3.0 to Web 4.0 technology advocated by the World Wide Web Consortium (W3C)[1)].

1.1 History

The study of the history of Software Engineering (SE) and its methods is beyond the scope of this work. However, to understand the development of science and technology, one needs to have an overview of its past to understand the context and to evolve and further develop it. See Appendix 1.

1.2 Criteria used

The comparison metrics applied (as described in Section 6) are designed to be self-contained and approachable. There are numerous metrics to be taken into account in the scientific evaluation of any methodology, however the specified criteria in this work is much clearer to all software experts or users of these approaches.

[1)] W3C is an international community where member organizations, full-time staff and the contributing public work together to develop Web standards led by Web inventor Tim Berners-Lee and Jeffrey Jaffe to lead the Web to its full potential. http://www.w3.org/

It is intended that this paper will act as a catalyst to new research in the field of software engineering. It is hoped that the results will lead the ICT community to understand the current shortcomings of existing approaches and thus promote the development of new innovative, integrative and fast software development techniques, which *should be* suitable to being used in all other fast-growing domains of computer world.

1.3 Paper outline

This paper is composed as follows: Sections 2, 3, 4 and 5 discuss the basics of the four methodologies respectively: Agile Software Development, Component-Based Software Engineering, Aspect-Oriented Software Development and Mashups. Each methodology is introduced by specifying the most important characteristics, advantages, disadvantages and their practical applicability. Section 6 gives a comparative analytical discussion based on several metrics that are important to researchers and developers. Finally, in Section 7 the overall discussion and conclusion is presented, with indications of future research work that would possibly interest enthusiast researchers, practitioners and developers in the software development business as we move from Web 3.0 to Web 4.0 technology and beyond.

2 Agile Methods

2.1 Agile methods definition

The problems of the conventional Waterfall Model, such as the low changeability of requirements, prompted developers to adopt the Spiral Model as a more dynamic and interactive design and development model [2]. However, the users still needed the opportunity to be able to change a system's requirements specification when necessary, leading to faulty system and code production and delivery time overruns. To overcome such problems, new software development approaches and techniques were introduced, Agile methods being one of them. It attempts to involve the customer actively in the software development process life-cycle as a key driver in order to ensure that all user requirements are considered.

The basic goals of introducing "Agile methods" in software engineering by Martin Fowler, Jim Highsmith and fifteen other developers [3] culminated in:

- *Individuals and interactions* over processes and tools,
- *Working software* over comprehensive documentation,
- *Customer collaboration* over contract negotiation,
- *Responding to change* over following a plan.

They summed-up their perspective, saying that the "Agile movement is not anti-methodology. In fact, many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modelling, but not in order to file some diagrams in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely used tomes. We plan, but recognize the limits of planning in a turbulent environment" [3].

As set out above, Agile methods use an iterative approach to specify, deploy and deliver the software and to support software development while responding well to requirements' rapid change (especially in the development process). Therefore, by delivering the working *portions* of the software quickly to the customer, he/she can be part of this process immediately and recommend new requirements which can be included in the further iterations of the system. This will finally produce a fast-finished overhead-reduced software development process.

In brief, enabling the agility of an organization in its software development working methods is the fundamental purpose of Agile methods, which implies quick delivery and accommodates changes quickly and as often as they are required [4-5]. The proviso is to react to changing circumstances in a timely and an orderly manner.

2.2 Agile methods characteristics

Agile methods are designed to have the following common characteristics:

- *Incremental:* In accordance with the requirements and based on functionality, small subsystems or increments with minimal-planning strategy are allocated from the system. Each new release represents a new functionality.
- *Iterative:* A full system is delivered over at first and then the functionality of each subsystem changes with each new version. The "timebox" for each iteration lasts from one to four weeks. Each timebox consists of a full software development cycle, which is planning, requirements capture, system design, coding, testing, acceptance and demonstration.
- *Emergent:* This characteristic addresses the smooth adoption of technology and requirements within the cycle of software production.
- *Self-organizing:* While having a cross-functional composition, the team is capable of perfectly completing the work items through self-organization. Team members are typically responsible for tasks to meet the functionality requirements.
- *Face-to-face communication:* The team members are strongly supposed to have daily meetings. Even if they are not in one place, they should keep contact via voice or video conferencing.
- *Single open office:* The team (5-9 people) works in a so called "bullpen" to communicate and collaborate easily.
- *Customer representative:* No matter what disciplines rule the team, a representative is appointed to be part of the team on behalf of the customer to answer the developers' mid-iteration questions.
- *Working software:* Primarily, the software should work. This is the first measure to evaluate the work, and along with the face-to-face communication, the customer's satisfaction will increase. As a result, the documentation will have less importance and will be decreased.
- *Tools and techniques:* Along with the particular tools, production optimization techniques are applied to improve project quality and agility [6].

2.3 Agile methods advantages

Mary Poppendieck [7] lists common key advantages of some Agile methods, which can be basically considered as

the same for all Agile methods. These advantages together with some other practical results from [8] are:

- Consumables such as diagrams, graphs and models that are invaluable to the final deliverable are removed or improved.
- The system is tested early and re-factored when necessary.
- Developers are briefed what to do but not given instructions of how to do it.
- Customers' demands are fulfilled due to their close interaction with the developing team which allows them to change their minds.
- The use of iterative development decreases the total development cycle-time up to 75 %.
- The process improves by taking past experiences, mistakes and successes into account.
- The project inventory is minimized since intermediate accessories such as captured requirements and design stage documents are minimized.
- Adversarial relationships are avoided. Instead, work is only done in order to deliver the best software.
- Requirements support flexibility and are pulled from demand.
- The scopes can be flexibly managed.
- Work-loads are highly stable.
- A fixed number of developers develop a large-scale software system, according to the high utilization of work load.
- Project management and production plans are highly flexible to any change.
- Higher quality of software resulting from earlier feedback from the customers as a result of this close interaction [4, 7, 8].

2.4 Agile methods disadvantages

Despite all the advantages of Agile methods as an interesting pathway to software development, there are a number of disadvantages that should be recognized, namely:

- It is usually difficult for programmers to understand the cooperation of functionalities. A decent overview is required for complex systems in order to avoid such confusion.
- Customers generally show contentment over early deliverables but lose patience and begin to complain about many shortfalls such as redundancies and later iterations. The customer has to be coaxed into providing useful feedback early and honestly. Elssamadisy and Schalliol [9] maintain that it is necessary for the customers to realize that software product is like purchasing a tailor-made suit that requires several tryouts before being satisfied with the perfect suit.
- Contrary to what is commonly understood about ease of Agile methods, the story cards that are used for programming are generally incomplete. In fact, weeks of intensive work is needed to write the story cards with a complete story comprehensively in order to develop the associated program to a high standard. Further, the developers should come up with a checklist of tasks to not only finish the story, but also verify that they catch up with these rules: to recognize weakly estimated stories, as well as to reprioritize them.
- Agile methods usually lack good scalability since the approach is following an integrative methodology. Understanding where exactly the project stands is not easy.
- Also in management, they can handle only small and medium-sized teams, not large ones.
- Agile development requires a team of individuals with high skills and motivation, who are not always guaranteed to be available [5, 8].

2.5 Agile versus plan-driven

Comparing Agile and Plan Driven methods (see Table 1) will create a neat image of the differences between the two methodologies and clarify the principle concepts behind the novelty of Agile methods [10].

Table 1 A comparison of the principles of Agile and Plan-driven methods

Perspective	Agile methods	Plan-driven methods
Primary objective	Rapid value	High assurance
Developers	Agility Knowledgeability Collocation Collaboration	Plan-orientation Skills adequacy Accessing external knowledge
Customers	Dedication Knowledgeability Collocation Collaboration Representation Empowering	Accessing to knowledge Collaboration Representation Empowering
Requirements	Highly emerging Rapid changing	Knowable early Highly stable
Size	Smaller (teams and products)	Larger (teams and products)
Architecture	Design for current requirements	Design for current and predictable requirements
Refactoring	Inexpensive	Expensive

2.6 Agile in practice

Before and since the proposal of Agile Manifesto, several Agile methods have been introduced and developed, some of which are:

- *Extreme Programming (XP)* by Beck, 1999, 2000
- *SCRUM* by Schwaber and Beedle, 2001
- *Dynamic Systems Development Methodology (DSDM)* by Stapleton, 1997
- *Adaptive Software Development* by Highsmith, 2000
- *Lean Software Development (LSD)* by Poppendiecks, 2003
- *Agile Modeling (AM)* by Ambler, 2002
- *Crystal Methods* by Cockburn, 2001
- *Feature-Driven Development (FDD)* by Palmer and Felsing, 2002
- *Pragmatic Programming* by Hunt and Thomas, 1999.

Tab. 2 compares the prescriptive characteristics of the above mentioned methods.

2.7 Agile skills and trainings

A majority of the practitioners believe that Agile

Table 2 Prescriptive characteristics [5]

	Scrum	XP	Crystal	FDD	LSD, DSDM, AM
Iteration length	4 weeks	2 weeks	<4 month	<2 weeks	Criteria used in this table are not applicable to these methods according to their variety type/parameter nature
Team size	1–7	2–10	Variable	Variable	
System criticality	Adaptable	Adaptable	All types	Adaptable	
Distributed support	Adaptable	No	Yes	Adaptable	

methods demand less training than traditional methods [5]. Pair programming, for example, helps reduce the required training since the programmers train each other as they develop the program. However, this training approach also called tacit knowledge transfer is disputed by experts to be more functional and useful than the normal explicit training. In this way, the stress is on skill development instead of learning the Agile methods. Self-training is what occurs while training on how to apply Agile methods. The teams show to be training themselves successfully.

3 Component-Based Software Engineering (CBSE)

3.1 Component definition

Generally, a component is an independent software unit, creating a system in coordination with other components. However, some specialists define components from totally different perspectives. Councill and Heineman [11] focus on *standards* and believe that "a component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard". On the other hand, Szyperski [12] focuses on the *characteristics*: "a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". So, the priority of CBSE is the *separation of concerns* regarding the wide range of functionality available throughout a software system.

3.2 Component architecture and interactions

Components are defined by their "requires" (inputs) and "provides" (outputs) interfaces that are related by the function. These components are capable of providing a service to other client components, while they themselves could be clients as illustrated in Fig. 1. Each Interface specifies a one-way flow of dependencies from one module (which provides a service) to a client. Each flow implements usable services to the clients.

However, every related pair of module and client are co-dependent, which is, a client depends on the related module to receive a service in a particular method, and the module is dependent on the client to access and utilize the said services [13].

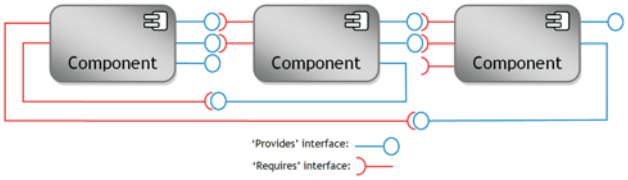

Figure 1 Component interfaces

1) *Provides:* This type of interface specifies the services that the component provides.
- This is the API of the component.
- Specifies the method that user can call.
- Denoted with a circle at the end of the line from the component.

2) *Requires:* This interface shows the services that should be provided in the system by other components.
- The component will not work if the service is unavailable.
- It does not compromise that the component will be independent or deployable.
- Denoted with a semi-circle at the end of a line to the component.

A component can also interact with other software entities, either component-based or traditional ones, via its interfaces. This interaction is possible with the whole system mounted on a component infrastructure as shown in Fig. 2.

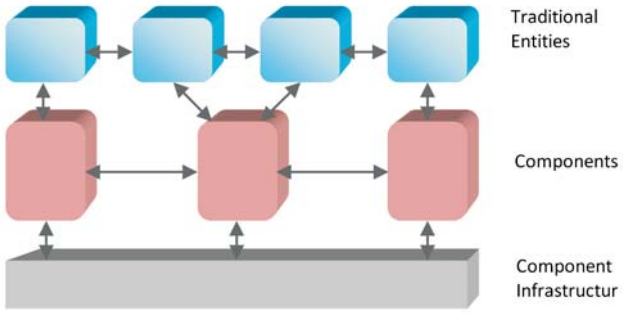

Figure 2 Component interactions

The detailed study of component's inner architecture is beyond the scope of this paper. Each component, based on the use, has several elements namely interface, port, connector, attachment, role, binding. Also depending on the applied CBSE model, the elements of the system may vary, some of which are: objects, properties, methods, events and facets.

3.3 Component characteristics

The most important technical characteristics of software components in a usage context are [14]:
- Assuming architectural embedding.
- Presenting each functionality via specific "incoming" or "provides" interfaces.
- Presenting parametric dependencies via "outgoing" or "requires" interfaces.
- Static dependencies.
- Targeting particular component platform.
- Requiring other components collaboration.
- Requiring per-instance context.

In brief, Ian Sommerville tabulates the initial characteristics of components as in Tab. 3.

Table 3 Component characteristics [15]

Component Characteristics	Description
Standardized	Component conforms to some standardized model which may define interfaces, metadata, documentation, composition and deployment.
Independent	Components can be deployed without using other particular components. When a component needs a service, it uses the "requires" interface explicitly.
Composable	All external interactions must take place through interfaces. Also a component must provide external access to information about itself, such as its methods and attributes.
Deployable	Component has to be able to operate as a standalone entity on the model platform. The component is binary and does not need to be compiled before being deployed.
Documented	Components have to be fully documented for the users to decide if they are satisfactory to their needs. All component interfaces should be syntactically and semantically explicit.
Testable	The interface should conform to specific standards. In this way the test approaches can also be standardized.

3.4 CBSE advantages

Modularization and the separation of functionalities as in CBSE have always been ideal for the software designers and developers. The advantages of using components are rather more fascinating than using objects. Recognized by domain experts, some of these advantages are [13, 16]:

- Better markets
- Confirmable license fees
- Encapsulation shielding or creating transparency
- Faster software delivery
- Software/hardware independence
- High functionality
- Immediate availability and early payback
- In-house software development
- Lower development and maintenance costs
- Lower risks of development
- Performance predictability improvement
- Reduced time-to-market
- Reusability
- Substitutability
- Tracking technology trends
- Upgrades regularly anticipate organization's needs.

3.5 CBSE disadvantages

The disadvantages of CBSE approach that users should take into consideration are:

- Constraints on functionality and efficiency
- Dependence on vendor
- Difficult to synchronize multiple-vendor upgrades
- Difficult scalability of the system
- High functionality compromises performance and usability
- Integration not always negligible due to some incompatibilities among vendors
- Less control over the growth and evolution of the system
- Less control over maintenance and upgrades
- Licensing delays
- Needed compromises in requirements
- Possible recurring maintenance fees
- Reliability of the system is often insufficient
- Up-front licensing fees.

3.6 Components versus objects

Very similar to the interrelated methods in object classes, components have several interfaces but that is not the only distinctive difference between an object and a component. As described by Sommerville[15], the components are usually defined similarly as object-oriented approach but they are basically different from objects in the following important ways:

- *Components do not define types.* A class defines an abstract data type; objects are instances of that type. A component is not a template used to define an instance, it is an instance.
- *Components are deployable entities.* They are installed directly on an execution platform, not compiled into an application program. The attributes and methods defined in their interfaces are accessible by other components.
- *Components are standardized.* Components have to conform to some component model that constraints the implementation, not like object classes that you can implement in anyway.
- *Components are language-independent.* Although components are usually implemented using object-oriented languages e.g. Java, implementing them in non-object-oriented programming languages is also possible. This is while Object classes have to follow the rules of a particular object-oriented programming language and generally can only interoperate with other classes in that language.
- *Component implementations are opaque.* Components are often delivered as *binary units* so that the buyer does not have access to the implementation. They are, at least in principle, completely defined by their interface specification. The total implementation is not visible to component users.

3.7 CBSE in practice

There are many implemented CBSE models, such as:

- *Component Object Model (COM)*: Introduced in 1993 by Microsoft, COM is a binary-interface standard for CBSE [17]. Dynamic object creation and inter-process communication in many programming languages are enabled by using COM.
- *Object Linking and Embedding (OLE):* Linking to and embedding documents and other objects developed by Microsoft are allowed by OLE technology. User interface elements can be developed and used by developers through OLE Control eXtension (OCX).
- *Common Object Request Broker Architecture (CORBA):*Produced by the Object Management Group (OMG) [18], and serving as a standard, CORBA enables the collaboration of components which are built using different languages and run on different computers. This, as a matter of fact, is the support for different platforms.
- *SOFA 2:* Developed by Distributed Systems Research Group at Charles University in Prague, SOFA 2 is a component system which provides several advanced features: behavior specification and verification using behavior protocols, ADL-based design, and connectors

supporting and using various communication styles, and finally enabling application transparent distribution.

4
Aspect Oriented Software Development
4.1
Aspect definition

Nowadays, there are many large scale, complex, distributed systems, all of them dealing with many *concerns,* such as security, auditing, logging, consistency, synchronization, error handling, timing constraints, user-interface and etc. These *concerns* that affect each other are called *crosscutting concerns* and due to their interrelationships cannot be well modularized using Object Oriented languages. The original idea of AOSD is to modularize these *crosscutting concerns.* Two main problems caused by *crosscutting concerns* are:

- *Code scattering* (one concern in many modules): This is when a concern exists in many places and leads to repeated code in the program.
- *Code tangling* (one module with many concerns): This is when a unit of decomposition carries out multiple tasks making it difficult to see what it is exactly doing.

The basis for AOSD technique was first presented by XEROX Corp [19] who found several programming and software developmental process problems where both procedural and object-oriented programming techniques were weak to sufficiently and clearly capture many of the major design decisions that the program must implement. This was the reason for those design decisions to be implemented in a scattered way throughout the code, resulting in tangled code, which was excessively tedious to develop and maintain. Hence, AOSD assisted by including the appropriate isolation followed by composition and reuse of the specified aspect code in order to express the programs involving such aspects with clarity.

"AOSD is an approach to software development that combines generative and component–based development" [15]. Main concerns in a program are identified and implemented as aspects. They are then weaved into the appropriate places in the program by a weaver which is the compiler (or interpreter) of an Aspect language. Fig. 3 illustrates the conversion of an Object Oriented program to Aspect Oriented and how a concern is modularized.

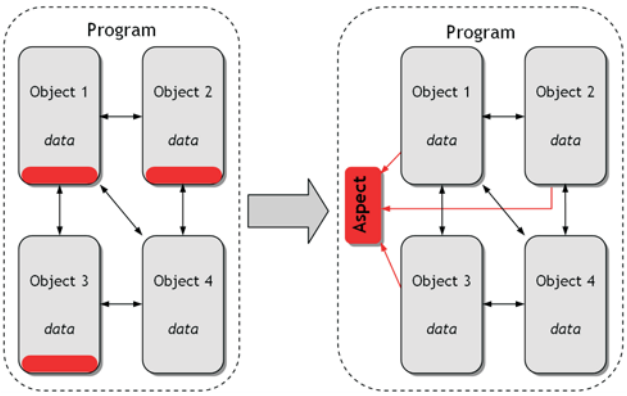

Figure 3 Modularizing system concerns

4.2
AOSD extensions

The result of emerging the idea of AOSD was the evolvement of several related concepts, all based on the Aspect-Oriented infrastructure, such as:

- *Aspect-Oriented Requirement Engineering:*
  Identify, specify and represent main properties at the requirement level. Examples: real-time constraints, security, and mobility. These mentioned properties broadly affect other architecture components.
- *Aspect-Oriented Modeling and Design:*
  Specify and characterize the system structure and behavior. Scattered and tangled *concerns* can be modularized.
- *Aspect-Oriented System Architecture:*
  Localize and specify main architectural level *concerns,* which cannot be modularized by using conventional architectural abstractions. Explicit mechanisms are proposed by specific system architecture languages to identify, specify and evaluate aspects at this level.
- Aspect-Oriented Programming (AOP):
  Specific programming tools and techniques to support modularization *concerns* during the implementation phase.

4.3
AOSD characteristics

As a young methodology for software development, AOSD has the following mentioned characteristics. The ones that are considered as advantages or disadvantages to this approach are listed in the following two sections.

- Aspects have a pre-definition of *where* to be included in a software system.
- AOSD allows the designers/implementers/users to understand each element of the system by knowing *only* its concern and without the need to understand other elements. Therefore, any required changes are localized to specific elements [15].
- *Quantification* allows the programmer to code the desired unitary statements separately, having effect on many non-local parts of the system**.** In other words, Aspects can crosscut any number of components, simultaneously.
- *Obliviousness* is that the abovementioned quantifications are applicable to any place in the system, not necessarily prepared for them as enhancements. Therefore, there is no need for the components to be aware of those aspects crosscutting them and also there is no need for extra effort from programmers to implement any functionality [20-21].
- *Aspect-Based dichotomy* is the distinction between components and aspects [20]. This dichotomy, along with separation of concerns and modularity, results in:
  - Decomposition of the system into components and Aspects.
  - Modularization of crosscutting concerns by Aspects.
  - Modularization of non-crosscutting concerns by components.
  - Explicit representation of Aspects, apart from other aspects and components.

4.4
AOSD advantages

The advantages of the application of AOSD that make it reasonable enough to be applied are [15, 22, 23]:

- Support for separation of concerns
- Solving scattering and tangling code problem
- A system concerns is treated in one place
- Changes are easily made
- Evolving requirements and functionalities can be easily adopted without major changes
- Support for on-demand computing (of configurable components)
- Capability of unifying the concerns (crosscutting) of non-functional features e.g. verification
- Support for code reuse
- Access control
- Modularity
- Uniformity
- Non-invasiveness
- Extensibility
- Transparency
- Reusability
- Flexibility, reusability and adaptability of all the elements that compose the system.

4.5
AOSD disadvantages

The main limitations and drawbacks of AOSD are:

- Young paradigm; there are as yet not enough proven tools and languages.
- Inspecting and deriving tests, which highly limit the usage of AOSD in large software projects.
- Lack of system test coverage tools to assess the dimensions of testing. This is because the aspects cannot be tested in isolation. They are tightly integrated with the base code of the system and may be woven into different places of the program with different reactions. To make sure, their function should be tested at any joint point of the program.
- Sequential top-down code reading is impossible. This makes Aspect-Oriented programs difficult for humans to understand [15, 22].

4.6
AOSD in practice

Although AOSD is a young paradigm and still under development, several major industrial projects have been implemented using AOSD concepts:

- *AspectJ:* Used by Sun Microsystems, AspectJ streamlines mobile application deployment for the Java ME. In order to simplify the mobile applications' development for different mobile-gaming community interfaces and operator decks, the use of Aspects seems inevitable.
- *IBM Websphere Application Server (WAS):* As a JAS (Java Application Server) with support for Web Services and Java EE, WAS is distributed in different editions, each of which supporting specific features. In order to isolate each edition's different features, it uses AspectJ internally.
- *Oracle TopLink:* As a Java object-to-relational persistence framework, and combined with the SAS (Spring Application Server), TopLink uses Spring AOP to achieve high levels of transparency.
- *JBoss Application Server (JBoss AS):* This is an open-source JAS with support for Java EE. JBoss AOP language is integrated in the core of JBoss AS and services such as security and transaction management are offered and deployed by the AOP.
- *Motorola wi4:* As a cellular infrastructure system, wi4 supports the WiMAX wireless broadband standard. The software that controls wi4 is implemented using WEAVER, which is an extension to the UML 2.0 standard and is used in the testing and debugging stages.
- *Siemens Soarian:* A health information system which supports access to the medical records of patients seamlessly as well as to the workflows definition for health providing organizations. Using AspectJ, Soarian integrates some crosscutting concerns and features.
- *ASML:* A lithography systems provider for the semiconductor industry.
- *Glassbox:* An agent for troubleshooting of Java applications, which supports the automatic diagnosis for common problems. The Glassbox inspector uses AspectJ to monitor the activities of the Java virtual machine.
- *.NET 3.5:* Supports the concepts of AOSD through the "unity container" which is a tool for dependency injection in order to boost the design and implementation stages as well as testing and administration of applications.
- *MySQL:* A widely used relational DBMS, the logging feature in which is totally implemented using AspectJ [24].

5
Mashup
5.1
Mashup definition

In Web development, a Mashup is a Web application, a Web page or even desktop application which performs a service by combining functionality or even data from two or more external sources. As its name implies, "Mashup" usually uses data sources and open APIs (Application Programming Interfaces) to follow a rapid and easy integrative approach to produce outputs that were not the principle goal of the system making the initial source data. As a very good model to understand what Mashups are, we can refer to the development history of existing computers. Current operating systems are using a combinative collection of *reusable components* each providing APIs for developers to build their desired applications and several interfaces as well, such as mouse and keyboard. Other possible APIs might be made to access the file system, display or network (see Fig. 4).

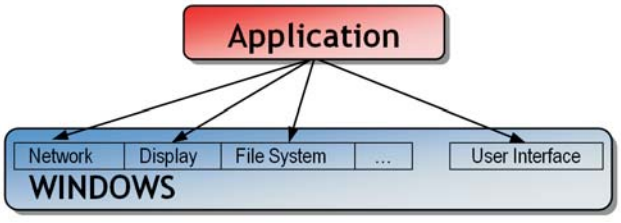

Figure 4 An application via APIs

As an analogy to the operating system concept, since the coming of the Internet, new generic APIs have been

introduced, allowing for rapid programming and development through the middleware which supports internetworking. Several companies are currently providing these APIs with the help of this middleware (see Fig. 5).

Google, Yahoo, Amazon and eBay [25-28] are good examples of these companies with their famous APIs, each of which are used to carry out the specified tasks, for example, following maps, reading news, sale and purchasing publications and ecommerce respectively. So, all of these companies put their specific APIs on the internet for the developers to access and combine in order to make theirs.

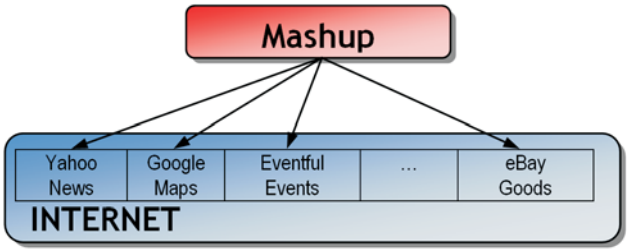

Figure 5 A Mashup via APIs

Assume that we want to make an API which, for example, informs us about the events that take place in our neighborhood area. Therefore, we access the Google Maps API and events database API, put them together and obtain a map that shows where each of the events is taking place in our zip postal code. In this case, the developer can take APIs from several Websites and merge or *mash* them together in order to innovatively create a remarkable new application that did not exist before on the Web; this is a Mashup. Today more APIs are created on the internet which results in development of more Mashups using these different APIs.

Mash Maker developed by the Intel Corporation is a good example of rapid advances in the Mashup field, which is intended for use by the ICT masses [29]. In terms of aesthetics and innovation, it is highly conceptual and creative as well as functionally radical in the new ways of browsing the Web through Web technologies and the supporting middleware. The Mash Maker philosophy is simple and catchy: its approach is presumed to be natural and simple. Therefore personalizing and creating "integrated meta application" using Mashup development are realized faster, more easily and are immediately usable. Mash Maker's built-in technology provides individuals with the possibility to browse the contexts, understand the semantics and visualize or present the information in the way they feel comfortable with, by controlling, tailoring and customizing the layout. That is made possible by allowing content from several sources such as Web, maps, photos, videos, RSS feeds, and displays to be combined in one place. It provides tools that let you manipulate, save items as favorites, edit, sort, merge, annotate and share Mashups with friends or with social network sites. This enabling technology in effect gives it the beauty to personalize and deliver information at an instant with the proviso of allowing people's areas of interests, preferences and fancy to be adapted.

5.2
Mashup architecture

There are two categories of Mashups from the architectural point of view:

- *Web-based:* Generally, the data are combined and reformatted on the user's Web browser.
- *Server-based:* The data are analyzed and reformatted on a remote server and then the final form output is transmitted to the user's browser.

5.3
Mashing up

Previously, the use of Mashups required the implementers to have an expertise in programming languages, specially Web programming, resulting in the construction process having five main procedural steps and shown on the left hand side in Fig. 6 [30]:

- *Data Retrieval:* Involves data extraction from Web sources like XML.
- *Source Modeling:* Assigning attribute name to each of data columns. Therefore existing data sources can be related to a new data source.
- *Data Cleaning:* Required to fix misspelling and transform extracted data into an appropriate format.
- *Data Integration:* Applied on extracted data to make it displayable.
- *Data Visualization:* The step of data display.

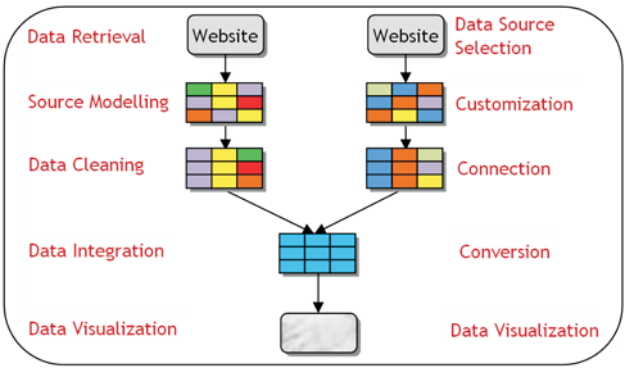

Figure 6 Steps in making Mashups

However, by virtue of newly released drag-and-drop Mashup tools such as Microsoft Popfly [31] and Yahoo! Pipes [32], these steps have been summed to *Data Source Selection, Customization, Connection, Conversion* and *Data Visualization* (as shown in Figure 6 on the right hand side) which eased Mashup programming into the realm of non-programmers. For ease of access to data and fast computation, transformation from the specific to the generic and vice versa is necessary when handling data in a multi-mode service-oriented environment. We can add data from specific legacy systems, such as old databases or old versions of software, and *generalize* them to be generic such that they can be readily used by all applications without further conversion. The opposite is also true. This reduces data access and computation time.

5.4
Mashup production process

Fig. 7 illustrates the primary steps in Mashup production. Data flow and control flow are indicated by dotted arrows and solid arrows respectively. Also the process steps (or developer activities) are shown by the smooth edged rectangular boxes.

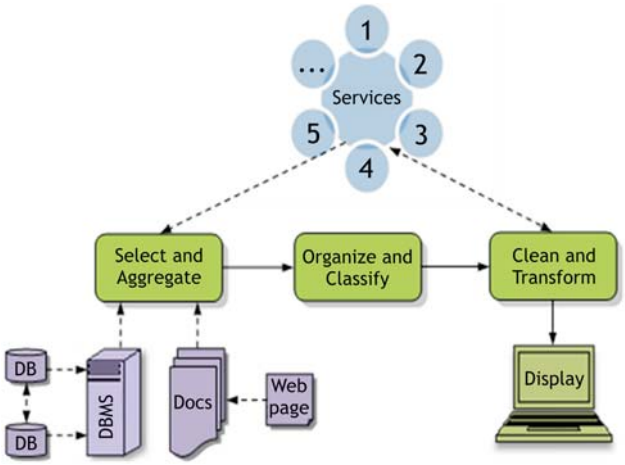

Figure 7 Mashup production process [33]

5.5 Mashup themes

Mashups usually fall under two themes due to research and practice:

- *Information systems (IS):* Where they are built to meet a particular need such as in:
  - Enterprise information integration
  - Map synthesis
  - Web service tracking.
- *End-User Programming (EUP):* Where end-user programmers (non-professional programmers) are able to use programming techniques to accomplish their task goals. Flexibility of the provided support is the most important issue in EUP systems. Since customization is unavoidable for an end-user to complete the desired task, therefore Mashups, due to their data-customization nature, are considered as a type of Web EUP [34].

5.6 Mashup types

There are three types of Mashups:

- *Data Mashups:* Where identical types of information and media from different sources are combined into a single representation.
- *Consumer Mashups:* Which targets the general public, Consumer Mashups are the most commonly used type of Mashup. This type of Mashups, unlike the data Mashups, can combine different data types.
- *Enterprise Mashups:* Enterprise Mashups gather the data into a single unique presentation and allow for a collaborative action between developers and businesses. They are visually rich and secure Web applications, exposing actionable information from various internal and external sources. According to the collaboration possibilities between developers and customer proxy (in order to define business requirements), Enterprise Mashups are suitable for *Agile Development* projects.

5.7 Mashup advantages

The block-based nature of making Mashups has the following advantages for the users of this technology:

- Existing applications are made possible to be reused with the help of Mashups.
- High decrease of the associated application development costs.
- Professional IT skills are not necessarily required to develop Mashups.
- Rapid application development is made possible.
- No approval is required from anyone if a user wants to add an API to the internet.
- Users' needs can be perfectly met by applications since the users are now able to incorporate content that they could not develop by themselves due to resource or time constraints.

5.8 Mashup disadvantages

As with any other system development approach, making Mashups has some important drawbacks:

- Although the content source is usually reliable, the problem of scalability potentially exists.
- Mashup is supposed to only include Web browsing software. Therefore, desktop applications hardly incorporate with it.
- No standard has been established for Mashup yet, and therefore it is very difficult to create and carry out security mechanisms.
- Security is another issue of these contents, particularly for businesses with sensitive data. They need to be assured that the information they are incorporating is not vulnerable.
- Since service-oriented architecture (SOA) has yet to become the basis for building most data sources, it is not easy to draw in the information. Mashups can be created in the absence of SOA; however they are highly facilitated by it.
- The content integrity is not constantly guaranteed.
- User's control over content quality and features is not guaranteed, as well as the constant support of the owner of the Mashup service or API.

5.9 Mashups versus portals

"Content aggregation" is the principle goal of both Mashups and Portals. As an older technology, portals are basically considered as an extension to conventional dynamic Web applications where data content is converted into a marked-up Web page after passing two phases:

1. Generating markup fragments.
2. Aggregating the fragments into Web pages.

Although Mashups and Portals have content aggregation as their common principle goal, they have several contrasting features as shown in Table 4 which has been adapted from [35].

5.10 Mashup in practice

According to the low level of required skills in order to make a Mashup as well as its in-house development characteristic, a huge number of Mashups have been made to date. The followings are some famous Mashup making tools in practice with a brief introduction, purpose and approach of each:

- *Yahoo! Pipes:* A hosted service of Mashups tool which provides a GUI for building data Mashups in order to aggregate Web pages, Web feeds, and many other services.
- *Microsoft Popfly:* A Mashup tool which is implemented as a visual programming environment in order to allow end-users to develop their desired Mashups by only connecting several blocks.
- *Marmite:* A Mashup tool on a workflow basis that helps users extract data from the Web and create applications with the use of a dataflow architecture in which data is processed by some operators, which are similar to Unix pipes.
- *Mashmaker:* A downloaded program which integrates into the Firefox browser by simplifying the *on-the-fly* approach to create Mashups where contents from several sources such as maps, RSS feeds, videos, photos and any other Web content are combined and displayed in one place.
- *IBM Mashup Center:* Written with the non-developer in mind, its design objective is to increase the number of its users while limiting the complexity of what can be built.

Some other tools to be named are: *WaveMaker Studio, Karma*, *Vegemite*, *iGoogle*, *OpenKapow*, *AboveAllStudio*, *Dapper*, *JackBuilder*, *aRex*, and *RSSBus*.

Table 4 Mashups versus Portals

Criteria	Portal	Mashup
Classification	Older technology. Extension to traditional Web server model.	Using newer, loosely defined "Web 2.0" techniques
Philosophy/ Approach	Aggregation is done by dividing role of Web server into two phases: markup generation and aggregation of markup fragments.	Using APIs (Application Programming Interfaces) from different content sites to aggregate and reuse the content in another way.
Content dependencies	Aggregates presentation-oriented markup fragments such as HTML, WML (Website Meta Language), VXML (VoiceXML), etc.	Can operate on pure XML content and also on presentation-oriented content such as HTML.
Location dependencies	Traditionally, content aggregation takes place on the server.	Content aggregation takes place either on the server or on the client.
Aggregation style	"Salad bar" style: Aggregated content is presented "side-by-side" without overlaps.	"Melting Pot" style: Individual content may be combined in any manner, resulting in arbitrarily structured hybrid content.
Event model	Read and update event models are defined via a specific portlet API.	CRUD operations (the major relational DB operations: Create, Read, Update and Delete) are based on REST (Representational State Transfer) architectural principles, but no formal API exists.
Relevant standards	Portlet behavior is governed by standards JSR 168, JSR 286 (Java Specification Requests: Portlet Specification 1.0 & 2.0) and WSRP (Web Services for Remote Portlets), although portal page layout and portal functionality are undefined and vendor-specific.	Base standards are XML interchanged as REST or Web Services. RSS (Really Simple Syndication) and Atom (Atom Syndication Format & Atom Publishing Protocol) are commonly used. More specific Mashup standards are expected to emerge.

5.11 Historical timeline of mashups

Fig. 8 demonstrates the history of Mashup tools based on a timeline [36-37]. As observed from this figure, while the momentum of Mashups has subsided to some extent since 2007, market share consolidation has taken place between 2008 to date.

Figure 8 Development of Mashup tools timeline

6 Comparisons

The processes of the discussed approaches are all traditional steps that are taken in software engineering. These actions are carried out by a human, or in some cases by "automation" tools, such as CASE or its extensions [38]. Now we will compare the four software development approaches from the human elements involved in the life-cycle of each, in terms of the following:

- Requirements capture
- System design specification
- Programming, testing & debugging
- System (program) integration
- Documentation
- Program maintenance.

Tab. 6 merely summarizes the result of the metric comparison between the four approaches. As can be seen from this table, there is a huge number of metric attributes or parameters involved in software development life cycles. We use the following terminology and grading scheme in compiling the table:

- The grades *'Very Easy'*, *'Easy'*, *'Medium'*, *'Hard'* and *'Very Hard'* refer to metrics related to required human effort, both physically and mentally. Not in all cases all of these grades appear but their ranking remains the same as stated.
- The grades *'Very Low'*, *'Low'*, *'Medium'*, *'High',* and *'Very High'* refer to metrics that indicate the quality, quantity or possibility.
- The grades *'Yes'* and *'No'* refer to metrics that indicate a possibility, occurrence or existence.
- The grade *'N/A'* indicates *'Not Applicable'*.

Tab. 5 is the key to the terms used in Tab. 6, giving the value of the weights in a percentile format. Note that some metrics are beneficial, while the others are known to be risky. These two types of metrics are shown as different categories in Tab. 6 and should be interpreted differently according to their relevant keys given in Tab. 5. Also note that, although a 'Yes' could be interpreted to be worth 100 points in some cases, according to the comparative nature of Tab. 6, 10 suffices as its weight. An example of this is

"predictability" which might have a range of possibility but we preferred to give it a Yes/No weight.

Table 5 Weights used for keys in Tab. 6

Benefits (Terms used)	Percentile points	Risks (Terms used)
Very Easy, Very High	100	Very Easy, Very Low
Easy, High	80	Easy, Low
Medium	60	Medium
Hard, Low	40	Hard, High
Very Hard, Very Low	20	Very Hard, Very High
Yes	10	Yes
No	5	No
N/A	0	N/A

The total weight of each software development method is calculated and shown in the last row of Tab. 6. CBSE seems to have the most weight compared to the others, and therefore we assume that it has the highest uptake in usage, which appears to be the trend at present. Agile methods still seem to need more concentration and improvement since they have the lowest total weight compared to the others. In between, come the AOSD and Mashup competing each other in benefits and drawbacks but highly different in the development process cycle. The maximum possible total weight for each method is 3600. Fig. 9 gives a graphical representation of the results of this study.

Although Mashup is evolving to become an "integrator", it still has several insufficiencies that cannot be ignored. This is because in any aspect of computer science, the approaches that are undocumented and need low level of skills have always been debated. Substitutability is another significant strength which the other three methodologies have, but Mashup lacks [39]. Not being licensed along with software dependency are other limitations that can however be overlooked by the impact of very high functionality. These issues must be dealt with, in the long run.

From Tab. 6 we select a few attributes to give a brief discussion of the comparison:

- ***Requirements inspection and capture:*** The inspection of system requirements is the most significant starting point of any software project and plays a key role in the assessment of any software development approach. Agile methods enable the dynamic capture of system requirement through the continuing involvement of the users. However, in the case of CBSE and AOSD, requirements should be best captured, especially for the input/output interfaces of the components at the outset. On the other hand, the intended simplicity of Mashup structures and their programming bestows the developers with the unnecessary time-consuming requirement capturing stage.
- ***Design quality:*** Agile methods contradict most of the pre-production principles of plan-driven methodologies. This is what makes a medium rated system design, in contrast with the modularized concepts of the other three approaches.
- ***Functionality:*** As described in the related section about Agile methods, the systems developed are supposed to include a new functionality with each increment. Therefore, the final system is presumed to cover all the desired functionalities. The modularized or function-based nature of CBSE gives this approach the support of a high range of functionalities, each of which is included in one of the components added to the whole body of the software. However, the separation of crosscutting concerns and the complexity of the systems built on AOSD concepts are what constraint the functionality of the resulted system. This kind of complexities and constraints do not appear in Mashups, and they usually seem to have a very high and close to expected functionality.
- ***Development speed and costs:*** The name says it all, Agile methods are used for speedy development. This is in contrast to CBSE, in which the components are designed and implemented before and are plugged into the system. That is what gives CBSE the highest development speed at a low cost. For AOSD, the

Table 6 Comparison of software development methods

Approaches	Agile Methods	CBSE	AOSD	Mashup
Metrics Benefits				
Requirements inspection & capture	Easy	Hard	Hard	Very Easy
Project management	Hard	Hard	Medium	Easy
Functionality	High	High	Medium	Very High
Design quality	Medium	High	High	High
In-house development	No	Yes	No	Yes
Development speed	Medium	High	Medium	High
Schedule-ability	Medium	High	High	N/A
Documentation	Yes	Yes	Yes	No
Testing (Basic)	Easy	Easy	Easy	Easy
Testing (Integrated)	Medium	Easy	Hard	Easy
Assessability	Medium	Easy	Hard	Easy
Maintainability	Medium	High	Medium	Low
Learning	Easy	Medium	Hard	Very Easy
Self contained	N/A	Yes	Yes	N/A
Structurability (Control flow)	Low	High	High	Medium
Structurability (Data)	Medium	High	High	High
Encapsulation	No	Yes	Yes	No
Modularity	Low	High	High	N/A
Reusability	Very Low	Very High	Medium	High
Substitutability	Medium	High	High	No
Predictability	No	Yes	Yes	Yes
Reliability	Medium	Medium	Medium	High
Efficiency	High	Medium	High	High
Integrity	N/A	High	Medium	High
Commonality	Low	High	High	High
Standardability	No	Yes	Yes	Yes
Interoperability	Medium	High	High	Medium
Portability	Low	High	High	High
Scalability	Very Low	Low	Medium	Medium
Self descriptiveness	Low	High	High	High
Expressiveness	Medium	Medium	High	High
Language independency	Yes	No	No	No
Augment-ability	Medium	High	High	Medium
Flexibility	Low	High	High	Medium
Composability	N/A	Yes	Yes	Yes
Satisfaction	High	High	High	Very High
Software independency	Yes	Yes	Yes	No
Hardware independency	Yes	Yes	Yes	Yes
Profitability	Medium	High	High	Low
Risks				
Development costs	Medium	Low	Low	Very Low
Maintenance costs	High	Low	Medium	Low
Complexity	Medium	Low	High	Low
Skill sets	Medium	High	Very High	Low
Risks	High	Low	Low	Low
Up-front licensing fees	N/A	Medium	High	N/A
Total Weight	**1820**	**2655**	**2210**	**2475**

weaving of the components and aspects seems to be more time consuming. In house development and no need for high skills are the hands to help Mashups to be developed fast and cheaply.

- ***Documentation:*** Principally, the Agile methods are supposed to have a very low amount of documentation. For CBSE and AOSD the document is initially for the description of components and their interfaces. However, after joining and weaving the part of the system, a new document should be released to explain the resulted functionalities. In Mashups, the document is not needed, unless there have been various interfaces and functionalities.
- ***Testing:*** Running the tests for block-based approaches are easier and usually of less complexity. However, testing the integrative systems of AOSD is a bit of challenge since the functionality of all the components and aspects should be tested singly as well as in all join points. For the Agile methods, the progressive addition of increments might result in the need for a complicated test scenario.
- ***Maintainability:*** Due to change of needs, the further changes into a software system are evidently inevitable. CBSE approach enables the system to become highly maintainable in terms of accepting changes. Again, for Agile methods, due to their incremental basis, maintainability can be a challenge. How a particular aspect of a system behaves in a specific join point can be a concern when making further changes in an AOSD system. Finally, Mashups are slightly maintainable due to their use of pre-implemented APIs or Mashups.

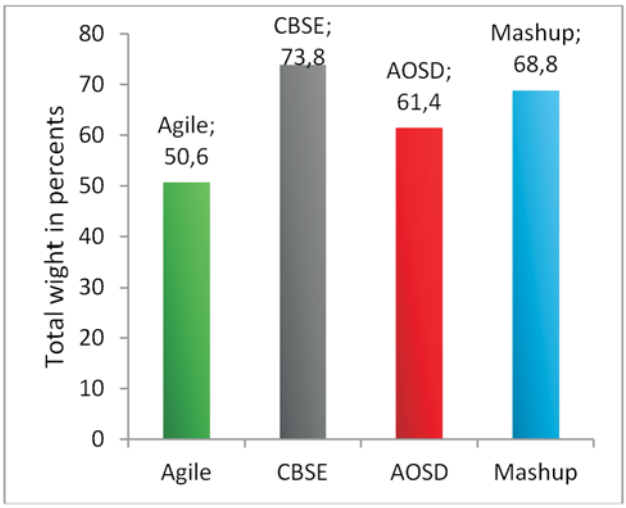

Figure 9 Total weight of software development methods in percentile format

7 Overall Discussion and Conclusions

7.1 Discussion

The concepts and characteristics presented in this paper are mainly conceptual in nature but they do have a strong influence on software development circles. In fact, accomplished software engineers in the field of software development around the world were impressed by the early ideas behind the Agile manifesto [40]. It acted like a magnet to return-to-origin, which resulted in a new research tide among visionaries, developers, scientist and researchers in the software industry, research laboratories and academia.

In summary, it can be observed that the software development "mind-map" is a complicated process in itself, with many factors involved in determining the definition, characteristics, advantages and disadvantages of each approach. The selection of any one approach is heavily dependent on the characteristics which embody that development and application domain which uses it. Thus, it will be fool-hardy to suggest that one approach is the "best" amongst them because of the differing characteristics and application domains. It is certain that selecting and generalizing one of these software development approaches as the most suitable one will be an impractical idea since each of them has so many potential advantages befitting particular applications and development methods.

However, undertaking this complicated comparison does indicate that given the latest technologies and tools for supporting domains like social networking and integrated user organizers over multiple heterogeneous resources and services, a new "all-encompassing" innovative approach would be an appropriate way to go. This might entail the combination of different approaches to synthesize the requirements and unify the ease and rapid delivery of the development.

7.2 Conclusion

We can conclude by stating that "one size does not fit all". Obviously, each of the methods discussed has its adherents as a function of the differing attributes of each approach. Only a cursory inspection of the costly elements such as training, stability, learning, etc. is all that is required to reveal that the initial development of a system based on any of the approaches is cheaper than its maintenance over time.

With the ever-growing need for faster development of applications and other systems software, there is the need to reduce the required programming skills by easing the programming paradigms. For instance, Mashups rely on data integration from different sources, requiring both Mashup tools to allow seamless integration and sufficient skills by end users as developers to easily make meta applications. But at present all the Mashup tools lack easy-to-use facilities and services, and the end users lack the necessary skills to successfully exploit Mashup development technology as a method. This is one of the main causes of discontinuation of some Mashup tools, for example Microsoft Popfly. End-user development as a paradigm is called e*nd-user programming,* also referred to as *Block-Based Programming(BBP)*, is beyond the scope of this paper but worth noting as an emerging technological solution, especially when it comes to discussing the commercial opportunities that are provided by Mashups.

We are currently undertaking research on block-based programming. BBP is based on the principles and dynamics of combining component-based programming approach with end-user programming paradigm that is viewed to be more complete and easier to use in practice. In this research study the new characteristics of Web 3.0 are considerably leaning towards more advanced software development methods. Web 3.0, where the computers are generating new information as well as the humans, embodies Mashup as one of its services. Web 4.0 with the development and evolution of the semantic Web are likely to demand better and smarted development approaches. This sort of integrated services, compared to those of Web 1.0 and 2.0, makes the new Web

technology more flexible and helps experts innovate and introduce new software engineering and development methods, as is the case with BBP.

Also, the idea of introducing a new concept called "outsourcing maintenance and management" seems interesting; the evolution of a system with the characteristic of "self-producing" code from a formal method. The next idea can be possibly providing aids for "self-testing", "self-documenting" and "self-maintaining", thus "self-managing" of the entire software life-cycle. These concepts should be all based on "autonomic computing" fundamental principles.

Although there are already some early indicative combinative models, such as Agile Mashup and Mashup as a Service (MaaS), the opportunities of conceptualizing and designing new approaches are still open. It also appears that MaaS will most probably be overtaken by new forms of application and data integration as well as "curried" Mashup with much richer gravy. It should also be noted that for example, by using Aspect-Oriented Programming, Components can be made and put together to make APIs for richer Mashups that are most suitable to be put into practice in the Agile methodology domain. Studying these possibilities is an exciting research challenge for future R&D work.

8
References

[1] Patel, A.; Seyfi, A.; Tew, Y.; Jaradat, A. Comparative Study & Review of Grid, Cloud, Utility Computing & Software as a Service for Use by Libraries. // Library Hi Tech News Journal, Print Published, 28, 3, 06-May-2011.

[2] Pressman, R. S. Software Engineering: A Practitioner's Approach, 6 ed., 2005.

[3] Beck, K. et al. Manifesto for Agile Software Development. Available: http://www.agilemanifesto.org/(2001, 20 November 2010)

[4] Highsmith, J. et al. Extreme Programming. // E-Business Application Delivery, ed, 2000, pp. 4-17.

[5] Cohen, D. et al. An Introduction to Agile Methods. // Advances in Computers, 62, ed: Elsevier, 2004, pp. 1-66. doi: 10.1016/S0065-2458(03)62001-2

[6] Lindvall, M. et al. Empirical Findings in Agile Methods, 2002, pp. 81-92. doi: 10.1007/3-540-45672-4_19

[7] Poppendieck, M. Lean programming, http://agilealliance.org/show/783 (2001, 28 September 2010)

[8] Sanjay, A. V. Overview of Agile Management & Development, http://www.projectperfect.com.au/info_agile_programming.php (2005, 25 October 2010).

[9] Elssamadisy, A.; Schalliol, G. Recognizing and Responding to "Bad Smells" in Extreme Programming. // Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, 2002.

[10] Boehm, B. Get Ready for Agile Methods, with Care, Computer, 35, (2002), pp. 64-69.

[11] Councill, B.; Heineman, G. T. Definition of a Software Component and Its Elements. // Component-Based Software Engineering: Putting the Pieces Together, ed Boston: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 5-19.

[12] Szyperski, C. Component Software: Beyond Object-Oriented Programming: Addison-Wesley Longman Publishing Co., Inc., 2002.

[13] Bachmann, F. et al. Volume II: Technical Concepts of Component-Based Software Engineering, Software Engineering Institute, Carnegie Mellon University CMU/SEI-2000-TR-008, 2000.

[14] Szyperski, C. Component technology: what, where, and how? // Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, 2003.

[15] Sommerville, I. Software Engineering, 8 ed, 2007.

[16] Boehm, B.; Abts, C. COTS Integration: Plug and Pray? // IEEE Computer, vol. 32, January 1999, pp. 135-138.

[17] Microsoft. COM: Component Object Model Technologies, http://www.microsoft.com/com (1993, 12 November 2010).

[18] O. M. Group. Common Object Request Broker Architecture (CORBA), http://www.omg.org/corba (1991, 12 November 2010).

[19] Kiczales, G. et al. Aspect-oriented programming, ed. 1997, pp. 220-242. doi: 10.1007/BFb0053381

[20] Chavez, C. v. F. G.; d. Lucena, C. J. P. A Theory of Aspects for Aspect-Oriented Software Development. // The 7th Brazilian Symposium on Software Engineering, 2003. doi: 10.1.1.94.2670

[21] Filman, R. E. What is Aspect-Oriented Programming, Revisited, 2001.

[22] Schwanninger, C. et al. Encapsulating Crosscutting Concerns in System Software. // Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software, 2004.

[23] Sharma, S.; Bhatia, R. Storage and Retrieval Using Aspect Oriented Technique. // National Conference on Challenges & Opportunities in Information Technology, 2007.

[24] Rashid, A. et al. Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. // Computer, vol. 43, 2010, pp. 19-26. doi: 10.1109/MC.2010.30

[25] Google. Google Maps, http://maps.google.com (2010, 11 September 2010).

[26] Yahoo. Yahoo News, http://news.yahoo.com (2010, 11 September 2010).

[27] Amazon. Amazon, http://www.amazon.com (2010, 11 September 2010).

[28] eBay. eBay, http://www.ebay.com (2010, 11 September 2010).

[29] Intel. Mashups for the Masses, http://mashmaker.intel.com/web/ (2010, 11 September 2010).

[30] Tuchinda, R. et al. Building Mashups by Example. // Proceedings of the 13th international conference on Intelligent user interfaces, Gran Canaria, Spain, 2008. doi: 10.1145/1378773.1378792

[31] Microsoft. Microsoft Popfly, http://www.popfly.com/(2010, 11 September 2010).

[32] Yahoo. Yahoo Pipes, http://pipes.yahoo.com/pipes/ (2010, 11 September 2010).

[33] Murthy, S. et al. Mash-o-matic. // Proceedings of the 2006 ACM symposium on Document engineering, Amsterdam, The Netherlands, 2006, pp. 205-214. doi: 10.1145/1166160.1166214

[34] Zang, N. et al. Mashups: Who? What? Why?. // The CHI '08 extended abstracts on Human factors in computing systems, Florence, Italy, 2008. doi: 10.1145/1358628.1358826

[35] Wikipedia. Mashup, http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid) (2010, 11 September 2010).

[36] Keene, C. Five Free Mashup Tools You Should Know About, http://web2.sys-con.com/node/955886 (2009, 8 October 2010).

[37] Hinchcliffe, D. Assembling great software: A round-up of eight mashup tools, http://blogs.zdnet.com/Hinchcliffe/?p=63 (2006, 11 September 2010).

[38] Wikipedia. Computer Aided Software Engineering, http://en.wikipedia.org/wiki/Computer-aided_software_engineering (2010, 11 September 2010).

[39] Patel, A. et al. A Study of Mashup as a Software Application Development Technique with Examples from an End-user Programming Perspective. // Journal of Computer Science, 6, (2010), pp. 1406-1415, doi: 10.3844/jcssp.2010.1406.1415.

[40] Boehm, B. A view of 20th and 21st century software engineering. // Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006. doi: 10.1145/1134285.1134288.

[41] Seyfi, A.; Patel, A. Briefly Introduced and Comparatively Analysed: Agile SD, Component-Based SE, Aspect-Oriented SD and Mashups. // Information Technology (ITSim), International Symposium in, Kuala Lumpur, Malaysia, 2010, pp. 977-982. doi: 10.1109/ITSIM.2010.5561582
[42] Wirth, N. A Brief History of Software Engineering. // Annals of the History of Computing, IEEE, vol. 30, (2008), pp. 32-39.
[43] Northover, M. et al. Towards a Philosophy of Software Development: 40 Years after the Birth of Software Engineering. // Journal for General Philosophy of Science, vol. 39, (2008), pp. 85-113. doi: 10.1007/s10838-008-9068-7
[44] Yau, S. S. Position Statement: Advances and Challenges of Software Engineering. // Computer Software and Applications, COMPSAC '08, 32nd Annual IEEE International, 2008, pp. 1-9, doi: 10.1109/COMPSAC.2008.240

APPENDIX 1

A brief bullet-form history of software engineering is given in this Appendix as a snapshot to aid the novice reader:

1950's: A period essential to the era of computing:

- Research institutions and universities began to have large computers available to be primarily used in engineering, natural sciences and business.
- 1956 was the start point for establishment of requirements-driven processes. U. S. and Canadian air defense developed SAGE (Semi-Automated Ground Environment) as the first information processing project of the 1950's (see Figure 10).
- In 1957 IBM developed FORTRAN, the first known high level language.
- In 1958 the first ALGOL was developed, followed by its successor in 1960.

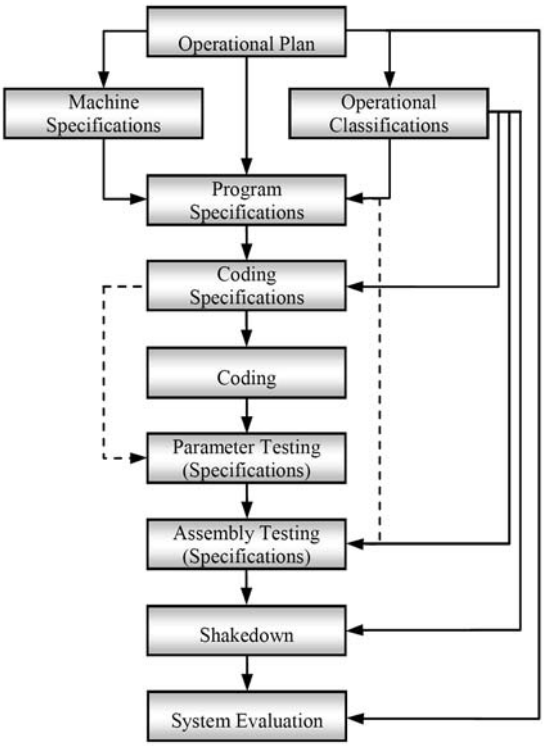

Figure 10 The SAGE software development process

1960's: Era of Code-and-fix (spaghetti coding):

- In 1962 the US Department of Defense specifically issued COBOL for business applications.
- In 1960's IBM developed PL/1 which was designed to unify the scientific and commercial worlds.
- In 1963, John McCarthy at MIT designed the first time-sharing system, adding interactivity to batch-processing systems.
- The term *"programming"* was commonly used through the mid-1960s and referred essentially to the task of coding a computer.
- In 1965 Dijkstra and Hoare published their papers on Structured Programming and Data Structuring declaring the programming to be a discipline, not a craft. Their ideas highly influenced new programming languages, particularly Pascal.
- The term *"software engineering",* referring to the disciplined & systematic approach to development & maintenance of software, was born at a NATO-sponsored Garmisch conference in 1968.
- Dijkstra (who presented at the Garmisch conference in 1968) identified an important source of trouble in the use of GOTO commands.
- Other trends:
  - Much better infrastructure; non-mathematicians could enter the field easily.
  - Although resulting in hard-to-maintain spaghetti code, manageable small applications were produced.
  - Universities established departments of computer science and informatics, emphasizing on software.
  - Software development and product companies were established.
  - Increased the number of large and mission-oriented applications.

1970's: Reacting to the 1960's code-and-fix approach:

- In 1972 David Parnas and Barbara Liskov introduced the concepts of information hiding and abstract data types separately, resulting in the concept *"modularization"* which is known to constitute the most important contribution to SE.
- Modern computing era started with the Alto in 1975 built by the Xerox PARC lab.
- In 1979 Modula-2 was introduced based on the modularization principles.
- Other trends:
  - Formal and Waterfall process models.
  - More carefully and well organized coding, after design and requirements engineering.
  - Adding the concept of confining iterations to successive phases.
  - The in-depth analysis of "people factors" in computer programming.

1980's: Emergence of Object-Oriented concept:

- In 1980 Smalltalk was developed, followed by Object-Pascal in 1985.
- In 1985 C++ was developed, followed by Oberon in 1988.
- Other trends:
  - Improvement of software engineering productivity and scalability.
  - Approaches to improve the productivity such as object orientation, very high level languages,

expert systems, visual programming and powerful workstations.

1990's: Concurrent and Sequential Processes:

- Java, a revolution in the Object-Oriented programming languages, was introduced by Sun Microsystems in 1995.
- During the late 1990's several Agile methods emerged, such as Crystal, Adaptive Software Development, Dynamic Systems Development, Feature Driven Development, eXtreme Programming (XP), and Scrum.
- Other trends:
  - Open source software development phenomenon started to spread.
  - The Spiral Model as a risk-driven process was intended to support CE (concurrent engineering).

2000's: Agility and Value:

- At 2000 Microsoft develops C#, with a vast support for Object-Orientation.
- Service Oriented Architecture (SOA) and Software as a Service (SaaS) are introduced as hot topics of software development and computing.
- The idea of the componentized software has been developed as in CBSE.
- Mashups, born in late 90's is known to be an integrative approach to software development is widely used. In 2005, World Wide Consortium (W3C) specified a standard language for representing ontologies on the Web built on RDF named OWL. The latest version, OWL 2, introduces profiles to improve scalability in typical applications. It is considered as the starting point of evolution of model-driven to ontology-driven development of software called Ontology-Driven Architecture for Software Engineering (ODASE) [40].
- Since 2010, major Mashup vendors have added support for hosted deployment based on Cloud Computing solutions.

Major advances in software engineering since 1960's:

- Software processes evolve from sequential, slow and rigid processes (such as Spiral or Waterfall processes) to incremental and iterative agile processes (such as eXtreme Programming).
- Software design and programming techniques, such as object-oriented programming and model-driven architecture.
- Different software-reuse techniques, such as component-based SD and design patterns.
- Numerous formal methods in order to specify, verify and test software systems.
- Software evolving architectural models, such as event-driven architecture, service-oriented architecture and ADL (architecture description languages) [41-44].

Authors' addresses

Prof. Dr. Ahmed Patel
School of Computer Science
Centre of Software Technology and Management
Faculty of Information Science and Technology (FTSM)
Universiti Kebangsaan Malaysia (UKM)
43600 UKM Bangi, Selangor Darul Ehsan, Malaysia

Mr. Ali Seyfi
School of Computer Science
Centre of Software Technology and Management
Faculty of Information Science and Technology (FTSM)
Universiti Kebangsaan Malaysia (UKM)
43600 UKM Bangi, Selangor Darul Ehsan, Malaysia

Miss. Mona Taghavi
Department of Computer Science and Research Branch
Islamic Azad University
Tehran, Iran

Miss. Liu Na
School of Computer Science
Centre of Software Technology and Management
Faculty of Information Science and Technology (FTSM)
Universiti Kebangsaan Malaysia (UKM)
43600 UKM Bangi, Selangor Darul Ehsan, Malaysia

Miss. Rodziah Latih
School of Computer Science
Centre of Software Technology and Management
Faculty of Information Science and Technology (FTSM)
Universiti Kebangsaan Malaysia (UKM)
43600 UKM Bangi, Selangor Darul Ehsan, Malaysia

Dr. Christopher Wills
School of Computing and Information Systems
Faculty of Science, Engineering and Computing
Kingston University
Kingston upon Thames KT1 2EE, United Kingdom

Prof. Dr. Sanjay Misra
Department of Computer Engineering
Atilim Unívers ity, Ankara, Turkey