Introducing an Agile Method for
Enterprise Mash-Up Component Development

Wolfgang Ketter
RSM Erasmus University
wketter@rsm.nl

Marko Banjanin
Westernacher & Partner Consulting B.V.B.A.
marko.banjanin@westernacher.com

Rob Guikers
Jibes B.V.
rguikers@jibes.nl

Alfred Kayser
Jibes B.V.
akayser@jibes.nl

Abstract

The introduction of an enterprise mash-up (EMU) development in an organization requires special consideration to balance the agile aspects with the control of a defined approach. In this article a new agile method for EMU component development is presented and defined, as a result of the cooperation between Erasmus University and Jibes B.V. This article describes the approach taken, the held survey and case study, and the resulting method Mash\Match. The Mash\Match method is a combination of aspects of four different agile methods: XP, DSDM, PP, and FDD. Mash\Match can be seen as new rising agile guideline upon which enterprises can effectively manage their business processes. Based on this result, Erasmus University and Jibes will further develop this method, by applying the results in practice and by further academic research.

1. Introduction

Gartner predicts that by 2010 more than 30% of the Global 2,000 companies will reconfigure their IT operations and enter an era of enterprise mash-up (EMU) adaptation. It is predicted that by 2010 more than 50% of Global 500 enterprises will re-engineer their core business processes based on web-based models [1]. This emphasizes the importance of Web 2.0 within the commercial world and implies that enterprises are leaning towards more agile and flatter organizational IT operations supported by new technologies that can influence new business opportunities.

This article describes the result of cooperation between Erasmus University and Jibes – Emerging Integration, being the definition of a new agile methodology specifically devoted for the purpose of development of EMU components. The approach taken was to compare different agile development methods and combine this with a survey held with experts in the field of agile development. This resulted in a selection of aspects of each method, of which the combination has then led to the definition of the Mash\Match method for agile EMU component development. The scope of this article and survey described in this article is limited to the project aspects of agile development, and does not describe the technical aspects that are generally related to the agile development practice.

2. Enterprise mash-ups

A mash-up application can be characterized as a lightweight and tactical presentation layer that uses the Web platform – Web-Oriented Architecture (WOA) [2] – in order to integrate multi-sourced applications into one web-based offering.

An enterprise mash-up (EMU) application is a mash-up application used to solve a specific real business problem with more emphasis placed on the tactical and strategic relevance, opportunistic, dynamic, and user-driven application development [3]. EMUs require enterprises to reconfigure their IT departments by placing emphasis on building scalable and reusable environment of mash-up components. Only then can EMUs be recognized as a corporate asset [4].

EMU application development can be performed by the end users, like normal mash-ups, but in this paper we focus on information specialists using this technology. Mash-up technology used in this way allows a very quickly develop of situational information applications combining existing information feeds, widgets, and services.
2.1 Business benefits

In the introduction to business benefits of using EMUs, we present a case study in which Westernacher & Partner Consulting solved a complex business scenario with an EMU application. The engagement involved a utility company running its operations in countries A and B that had a problem of having two decoupled systems where both were imperative for sound business transactions. These two separate systems, GIS and ERP, were facing the problem of having inaccurate and incomplete information on asset addresses and locations across the entire system landscape. This resulted in operational inefficiencies and unnecessary costs, such as sending crews to wrong places. Westernacher approached this issue by initially creating a master data management system to leverage both GIS and ERP systems and sync them on a master data level. On top of that, integration of SAP and a GIS services was established using SAP NetWeaver Process Integration (PI) composite (mash-up) application capabilities to capture data and trigger transactions in SAP. The results from mashing-up the two sources were displayed on Google maps. The result was an EMU application that facilitated instant information flows, providing reliable notifications, and sending a repair crew to the right address.

Following this example and the literature, we identify the following business benefits obtained from utilizing EMU applications [4, 5, 6]:

1. EMUs offer a less expensive and faster application development since mash-ups combine existing and reusable functionalities. This lowers the development time and increases application’s functional agility, as well as business process agility, and facilitates better management of application evolution.
2. EMUs have the capability of better information distribution since internal and external information flow is automated and conducted in real-time.
3. EMUs empower users to modify and customize applications to their own preferences.
4. EMUs enable enterprises to address the long tail of applications. By employing mash-up components IT departments can create a more scalable application delivery mode by means of having a wide access to various internal and external resources. Increase in application’s scalability can result in a significant increase in the development reach as well as in return on investment.
5. EMUs increase the scope of the way applications can add value to business processes. EMUs have this potential because they have the capability to automate work that was previously too complex to handle by a single application.

There are similarities between the business value of EMUs and Business Intelligence (BI) implementations, allowing the business to have more insight in their information. However, the fact that both serve a purpose in creating insight in information does not mean that they are the same. EMUs are positioned as long tail applications. They are short-lived (implying the need of constant life-cycle management since mash-ups evolve over time) and extremely dynamic, while the BI dashboards and reports are working with information from already aggregated data through data warehouses and Extract, Transform, and Load (ETL) tools. This means that information cannot be combined in real-time by an average business user. Further evolution of EMUs is expected to be delivered by a BI tool, but for now BI serves as just another source for EMUs.

2.2 Technology

The technology behind EMU platforms can be diverse but there are some key technologies which make it possible to develop and use EMUs in the short development cycle that they require. Awareness has to be placed on the fact that EMU creation is targeted at the business user instead of IT. Within EMUs three basic requirements need to be handled:

- Adapt to existing services and information;
- Aggregate data/information (server-side); and
- Present information (client-side).

To adapt to existing services and data these services and data sources need to be bridged to a simpler format which is human readable and ready for aggregation on the server side. JavaScript Simple Object Notation (JSON) and eXtensible Markup Language (XML), in the form of ATOM syndication format, are the most commonly used standards. Traditional Simple Object Access Protocol (SOAP) web services are bridged to REstpresentation State Transfer (REST) to ease the invocation of these services. The aggregation of information is simplified by the JSON and XML usage on the server-side of an EMU. Different technologies like Extensible Stylesheet Language Transformation (XSLT), XPath, and XQuery are already available to fill in this functionality. More important are the technologies used on the client-side of an EMU. Asynchronous JavaScript and XML (AJAX) and REST principles are used to create the rich user experience for a single output [7]. This will result in a so called widget which can be embedded in an existing web
The real benefit of these widgets will, however, be the ability to “talk” to other widgets. A sample of a standard is the iWidget standard [8]. By wrapping a widget in this standard the end-user can create messaging through JavaScript independent of the widget itself making it possible to create the client-side mash-up on the fly.

3. The scientific approach

With the main focus on finding the most suitable agile software project management methodology for EMU development, we compared nine most commonly used agile methodologies today. Our focus was turned towards the following agile methodologies, in the same order as in Table 1: 1) Adaptive Software Development (ASD) [9]; 2) Agile Modeling (AM) [10]; 3) Crystal Family [11]; 4) Dynamic Systems Development Method (DSDM) [12]; 5) eXtreme Programming (XP) [13]; 6) Feature-Driven Development [14]; 7) Internet-Speed Development (ISD) [15]; 8) Pragmatic Programming (PP) [16]; and Scrum [17].

To assess a method’s suitability towards EMUs we identified a set of 12 common factors within each agile methodology. The list was formed by surveying the literature and using a valuable more than 20 years of business experience in the field of software engineering from one of the author. The complete set of common factors is as the following:

1. Developers’ Skills (the level of developers’ understanding and conceptualization of different methodologies)
2. Project Management Support (supported stages during software development process)
3. Described Processes (overall inputs and outputs, as well as tools and techniques that are needed to successfully complete one project phase and proceed to another)
4. Guidance (extent of how a certain task can be executed by means of a set of practices and work products)
5. Programming (team structure around the development process)
6. Communication and Collaboration (between team members and projects stakeholders)
7. User Involvement (the level of involvement of end-users in the development)
8. Organizational Maturity (level of organizational maturity)
9. Culture (openness and flexibility of an organization)
10. Team Size (team size relevant to the development of EMU components)
11. Project Size
12. Development Time (in terms of how frequent iterations should be)

Thus, our conceptual model encompassed the comparison of nine agile software project management methodologies to 12 common factors. Initially we conducted an electronic survey study that encompassed the total of 55 industry professionals and academics. This electronic survey lasted full four months (June - October 2008). Secondly, we conducted a single case study and used it as an empirical tool to investigate a phenomenon of the development of EMUs within a real life context. The results of this survey are summarized in Table 1 (p.7) that gives a concise comparison of all nine agile methodologies on the 12 common factors of development methods. Shaded cells indicate building parts of the MashMatch methodology.

4. Agile development methods

The process of software development encompasses too many uncertainties; such as software complexity, pressure to constant changes [18], difficulties of delivering all functional requirements on time, customer relationships, and the development team relationship, just to name a few [19]. Various software development techniques have been used from 1970’s. Since then, times have changed and companies ranging from small size to enterprise size developed certain needs that would put those old practices behind and express needs for newer and more adaptable methodologies. For this reason, agile methodologies were thought of to accommodate fast changing requirements and at the same time boost firms’ operability, efficiency, and effectiveness.

Agility refers to flexibility, speed, responsiveness, collaboration, and change. Agility facilitates lightweight development process because of the empowerment of developers to make changes, involve customers in all stages of the development process to communicate with team members, to develop something in a speedy and simple way, develop in incremental life cycles, and at the bottom line it empowers them to better decision making [20].

5. New methodological approach

As a result of the investigation, this study presents a new methodology whose guidelines could be utilized
for the development of EMUs. This new agile methodology is coined as Mash|Match, and can be seen as a set of best practices taken from business proven agile methodologies. The following four agile methodologies create Mash|Match:

- **XP** - Responses indicated that XP can be seen as the main driver behind the development of EMU components. XP was recognized completely suitable in 9 out of 12 factors. The development team should utilize XP’s guidelines around the following agile factors: developers’ skills, project management support, described processes, guidance, organizational maturity, team size, project size, development time, and culture. For a more detailed description see Table 1 (p. 7).

- **PP** - The results clearly indicated that the most suitable development approach would be the one of PP agile methodology (teams built around the system’s functionality). This fact might be derived from the fact that mash-ups are combined of existing and reusable functionalities [4, 5, 6] and, therefore, need a more constructive approach of developing them. Also, to be able to practice such development, developers should have Level 3 of development skills (developers who posses qualities in that would enable them to tailor a chosen method to fit new situations) according to Alistair Cockburn scale [21].

- **DSDM** - Communication and collaboration are an essential building block of successful development of EMU components. Business users are required to openly communicate and collaborate with the development team in terms of what their needs are; and what kind of functionalities need to be developed, and by when. This approach can be completely utilized by DSDM’s MoSCoW (Must, Should, Could, and Won’t have) principle.

- **FDD** - FDD methodology claims that end-user participation in only necessary during the elicitation of functional requirements. Mash-ups are built based on user preferences and they should operate following user requirements so FDD provides an important step towards achieving the final goal.

Figure 1 depicts four methodologies combined into the Mash|Match method, by depicting which methodology is selected for which key area.

6. Applicability to real-life scenarios

Previous sections have described how the Mash|Match method has been developed based on comparison of different agile methods combined with the results of a survey held for this purpose. The resulting method is, therefore, a theoretical combination of different aspects of different methods, with some practical experience, collected via the survey. To explore the applicability of the Mash|Match method into existing real world situations, this section describes practical, current experience in the field, and describes how this method could be applied in typical larger and smaller development organisations.

6.1 Case study: supply chain organization

In 2008 Jibes was assigned by a subsidiary organisation of a large Original Equipment Manufacturer (OEM) in the semiconductor industry to provide a solution for their data integration issues between their main organisation and their local systems. The assignment included providing real-time information flow, customizing manufacturing information, finding a solution to eliminate mistakes in planning and production stages, and increase work productivity, efficiency, efficacy, and agility. Jibes selected EMU technology as the base of the solution to be realised.

The subsidiary and the main organization were exchanging excel sheets to provide the updated supply chain planning. These were manually analyzed for
changes that require adjusted planning in their own organization and their suppliers. By leveraging EMU technologies, Jibes managed to create a new IT environment within the subsidiary where the real-time flow of information was established between the OEM and the ERP system of the subsidiary. Jibes managed to apply EMU technology by connecting BI feeds with internal databases. With this approach Jibes enable to make forecasting and planning easier [22].

The approach that was applied by Jibes in this case study was based on their ‘Enterprise Mash-up Innovatory’ approach. This approach consists of the following steps:

1. **Mash-up Checklist** – Providing an initial check for assuring that the case can be implemented with a mash-up technology. This checklist will provide pointers and best-practices for positioning the solution as an EMU suitable for end-users. The checklist is a set of questions that test whether the situation and case is suitable for a EMU solution.

2. **Mash-up Foundation** – Providing a core check if data sources and services were available for the development of a mash-up. At this stage widget usage was investigated; as well as the needs for the end-user. This resulted in an advice on integration logic and widget development.

3. **Mash-up guidelines** – Assuring that the developer and end-user used the right standards to create the mash-up. This assured the maintainability and install-ability of the mash-up. These guidelines are derived from best practices and targeted at the end-user. For this reason, they will be tailored per project.

4. **Mash-up implementation** – This step included the real mash-up implementation. Depending on the outcome of the mash-up foundation, this results in an end-user mash-up, a development project or both for providing the data sources and widgets.

5. **Mash-up life-cycle management guidelines** – Because mash-ups are generally short-term applications these guidelines will make sure that the mash-up is monitored and recycled if necessary. This stage makes it possible for the IT management to get a grip on the ever expanding mash-up implementations and create a plan for optimal re-usability.

At the end of the project, the results and approach were evaluated and compared to the Mash|Match method to learn this method’s applicability to a real-life scenario, and to learn how the project could have benefitted from the Mash|Match method. The comparison resulted in the following key differences between the approach taken and the Mash|Match method:

1. Developers’ skills applied are higher than recommended by Mash|Match.
2. As the project size was small, consequently the team size is smaller than recommended by Mash|Match, and the organisation in to functional domains of the team did not apply.
3. Communication was on even more direct level then the Mash|Match prescribes, also due to the smaller team size.
4. Development time was significant shorter than the average development time predicted by Mash|Match, also due to the smaller nature of the case.

On most other aspects, however, the applied approach match the Mash|Match method, demonstrating that such an approach can be successfully applied in real world scenarios. The key success factor of this project was the quick delivery of results, mainly enabled by the use of combination of the new technology, and the use of agile development practices. Furthermore, the availability of a specific agile development for EMUs would have resulted in even quicker results, as part of the start-up of the project was spent on the development of a specific approach for this situation. Jibes has successfully applied the method for several smaller cases with great success. Total turnaround time was limited to weeks, and results were even achieved in a couple of days. Jibes expects that with new similar engagements the Mash|Match approach would enable to more quickly start EMU projects, as the approach provides the structure and methodology defining what is needed and what needs to be organized, alleviating the need to determine this for each project, as well as have more quickly agreement on the whole project, especially on the involvement of end-users.

### 6.2 Traditional enterprise IT development

Large organisations with large IT organisations, such as financial institutions and telecommunications, have generally rigid development organisations and methods, including rigid release and deployment methods. However, with the business trends of demanding more frequent and faster delivery of new functionality, agile development also becomes a necessity for such organisations [23]. The first step is to define explicitly in which area EMU technology will be used, and how it will be deployed in the organisation, as shown with the Westernacher implementation in section 2.1. Governance of EMU development is a
critical issue for enterprise (and other) organisations, as control of deployment, especially in terms of reliability and quality is a key factor.

Envisioned is that core systems (databases, applications) will remain in the traditional domain, but that for the agile development a separate organisation is to be defined, with a specific method for the development of EMU components. The Mash|Match method can be selected for this purpose, as it has been specifically designed for such situations. In this way the introduction of an agile development method is not interrupting the existing development organisation, keeping core system development separate from the more frequent changing frontend applications, that more and more will be based on EMU technology.

6.3 Smaller organizations

Organisations that do not necessarily have a full IT organisation with its own development organisation and corresponding rigid methods, rely more on vendors managing their core systems. These organisations typically have a small group of specialists taking care of the custom functionalities around the core systems. For example, a typical supply chain organisation has applications such as ERP and Customer Relationship Management (CRM) sourced from software vendors (that apply traditional development methods themselves), and face specific development requirements regarding data integration and presentation. In practice, this often results in data being exchanged as Excel sheets, with all its limitations, in terms of data management.

For these organisations, EMU technology can help overcome data integration and presentation issues. Typically, these organisations do not have the rigid development methods as the larger organisations have. Introduction of an agile method for EMU components is, therefore, even more suited for these kinds of organisations, providing the structure of methodology, whilst keeping the flexibility of agile development.

7. Conclusion

The Mash|Match method for agile development of EMU components has been defined on the basis of a comparison of twelve agile development methods, on the results from the survey amongst specialists in the agile development, and enforced by practical real-life experience of EMU development. We have shown that certain aspects of business proven agile methods can be utilized in the case of Mash|Match methodological approach. Four existing agile methods are suitable for the new methodological approach – Mash|Match, being XP, PP, FDD, and DSDM. The study indicated that XP has the greatest influence on Mash|Match (9 out of 12 factors) followed by PP FDD, and DSDM. Mash|Match method can be applied in both enterprise organisations with rigid development organisations, as well as in smaller organisations that have a more ad-hoc development approach. For the first, the method can be applied for specific front-end developments, and for the latter, to provide a methodology whilst maintaining the flexibility of ad-hoc development.

8. Future work

Further work will be needed to test Mash|Match methodology in real world scenarios to evaluate the benefits and learn its constraints. Since current work is focused on EMU component development we are planning to do a more complete research in this area. This new research will mainly focus on bringing the mash-up development life cycle into the picture. With the new research we can focus on providing answers such as “what kind of data governance organization needs to be established to ensure data quality when deploying EMUs within an organization”.
Table 1. Mapping of agile development methods to common factors.
The coloured cells indicate the Mash|Match selection.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Agile Method</th>
<th>ASD</th>
<th>AM</th>
<th>Crystal Family</th>
<th>DSDM</th>
<th>XP</th>
<th>FDD</th>
<th>ISD</th>
<th>PP</th>
<th>Scrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developers Skills</td>
<td>Requirements specification, design, code, unit test, integration test, system test, and acceptance test</td>
<td>L3</td>
<td>n/a</td>
<td>L2</td>
<td>L2 and L3</td>
<td>L1A and L2</td>
<td>L2 and L3</td>
<td>L3</td>
<td>L2 and L3</td>
<td>L3</td>
</tr>
<tr>
<td>Project Management Support</td>
<td>Requirements specification, design, code, unit test, integration test, system test, and acceptance test</td>
<td>Design, code, unit test, integration test, and system test</td>
<td>Project inception, requirements specification, design, code, unit test, integration test, system test, acceptance test, and system in use</td>
<td>Requirements specification, design, code, unit test, integration test, system test, acceptance test, and system in use</td>
<td>Project inception, requirements specification, design, code, unit test, integration test, system test, acceptance test, and system in use</td>
<td>Requirements specification, design, code, unit test, integration test, system test, acceptance test, and system in use</td>
<td>Requirements specification, design, code, unit test, integration test, system test, acceptance test, and system in use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Described Processes</td>
<td>Processes are not fully described</td>
<td>Processes are not fully described</td>
<td>Processes are fully described</td>
<td>Processes are fully described</td>
<td>Processes are not fully described</td>
<td>Processes are not fully described</td>
<td>Processes are not fully described</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guidance</td>
<td>Not concrete</td>
<td>Not concrete</td>
<td>Not concrete</td>
<td>Not concrete</td>
<td>Concrete</td>
<td>Not concrete</td>
<td>Not concrete</td>
<td>Concrete</td>
<td>Concrete</td>
<td></td>
</tr>
<tr>
<td>Programming</td>
<td>Developers, through JAD sessions</td>
<td>Test driven development principles</td>
<td>Several teams with 2 to 3 designers and programmers</td>
<td>Developers and senior developers</td>
<td>Pair-programming</td>
<td>Team of chief developers</td>
<td>n/a</td>
<td>Team programming around functionalities</td>
<td>In a team of programmers with assigned package</td>
<td></td>
</tr>
<tr>
<td>Communication and Collaboration</td>
<td>Constant communication between developers.</td>
<td>It has to be open and free. Emphasis on open communication.</td>
<td>In the same and shared space is a requirement</td>
<td>Encouraged by MoSCoW principles</td>
<td>It must be constant in the same environment</td>
<td>Highly depends on the teamwork where each team member can carry multiple roles</td>
<td>Through management-oriented framework</td>
<td>Team members must take responsibilities and take an active role</td>
<td>Team programming around functionalities</td>
<td></td>
</tr>
<tr>
<td>User Involvement</td>
<td>JAD sessions</td>
<td>Customers are always present</td>
<td>Direct user involvement where users give 2 user reviews per release</td>
<td>On-site where they share the same workplace and constantly evaluate working prototypes</td>
<td>On-site and they must be present at all time</td>
<td>To elicit functional requirements only</td>
<td>At the latest stage of the development process - process adjustment</td>
<td>Constant due to user-centred design</td>
<td>High user involvement during the development process</td>
<td></td>
</tr>
<tr>
<td>Organizational Maturity</td>
<td>2</td>
<td>n/a</td>
<td>2 to 3</td>
<td>2</td>
<td>2 to 3</td>
<td>2</td>
<td>n/a</td>
<td>n/a</td>
<td>2 to 3</td>
<td></td>
</tr>
<tr>
<td>Culture</td>
<td>Flexibility of management culture</td>
<td>Open and flexible</td>
<td>Flexibility from top management</td>
<td>Senior staff must empower developers to make rational decisions</td>
<td>Open, flexible, and supported by the management team and end customers</td>
<td>Open and honest environment</td>
<td>Has to value innovation and changes</td>
<td>Has cherish abstractions, new ways of thinking, and mutual understanding</td>
<td>Disciplined environment</td>
<td></td>
</tr>
<tr>
<td>Team Size</td>
<td>No built-in limitations</td>
<td>Small</td>
<td>6 to 40 developers in multiple groups</td>
<td>Between 3 and 20 developers</td>
<td>Several teams but small and dynamically formed</td>
<td>n/a</td>
<td>n/a</td>
<td>Between 3 and 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Size</td>
<td>Large size</td>
<td>No limits</td>
<td>Small to medium size</td>
<td>Very large</td>
<td>Medium size</td>
<td>Large size</td>
<td>n/a</td>
<td>n/a</td>
<td>Large size</td>
<td></td>
</tr>
<tr>
<td>Development Time</td>
<td>No built-in limitations</td>
<td>Within hours</td>
<td>1 to 4 months</td>
<td>2 to 3 years</td>
<td>Short iterations – up to 4 weeks per release</td>
<td>Iterations every 2 weeks</td>
<td>Short development cycles</td>
<td>n/a</td>
<td>30 day development cycles</td>
<td></td>
</tr>
</tbody>
</table>
9. References


