Towards an Agile Reference Architecture Method for Information Systems

Eric Souza
Universidade NOVA de Lisboa
er.souza@campus.unl.fct.pt

Ana Moreira
Universidade NOVA de Lisboa
amm@unl.fct.pt

Fernando Wanderley
Unicap
fernando@unicap.br

Abstract

Agility in software architecture development has received significant attention recently, but supporting tools and methods for this architecture-agility combination are still lacking. This paper proposes RAMA (Reference Architecture Modeling in an Agile software development), a value-centric method to address this issue. RAMA uses model-driven engineering to create information system's reference architecture aligned with the organization's business values. RAMA's feasibility was evaluated with a case study and a proof-of-concept tool.

1. Introduction

Traditional software architecture processes tend to introduce excessive documentation and additional development effort of possibly unneeded features [1]. This may be why combining software architecture design and agile development was ranked second in the top "ten burning research questions" for the agile community [2]. However, combining these topics is challenging [1], [3], as proved by a recent study [3] showing that the architecture-agility combination still lacks supporting techniques. The first challenge is the apparent mismatch between architectural design giving a development plan, and agile practitioners not paying much attention to planning and embracing changes during development [3]. The second challenge points to valuable information being lost or misunderstood due to communication issues between business and software developers, leading to wrong or needless architectural features [3].

This paper addresses these issues by proposing RAMA, a Reference Architecture modeling Method for information systems in the context of Agile software development. RAMA focuses on customer satisfaction, the core principle of the agile manifesto [4]. It creates an information system reference architecture aligned to the economic business values of an organization, and uses model-driven techniques to automate some development activities (e.g., generate source code). RAMA supports the creation of a software architecture model in an intuitive, interactive, and agile (fast) manner using a cognitive map (e.g., mind map). While mind map based models add simplicity (an important principle in the agile manifesto [4]) to the models, model-driven techniques are used to automate tasks required to generate a reference architecture. Model-driven tools were developed to check the viability of automating part of the RAMA's process and a case study shows RAMA's feasibility. The results indicate that RAMA is a good alternative to model software architecture for an information system in an agile context.

This paper is structured as follows. Section 2 introduces software architecture, agile practices, cognitive maps, business value, and model-driven engineering. Section 3 details RAMA and Section 4 evaluates it with proof-of-concept tools and an industrial case study. Section 5 presents related work and Section 6 concludes and identifies future work.

2. Background

2.1. Software architecture

Software architecture has been defined in many different ways [5], but at its core it refers to the structure of the software [6] comprising software elements, the externally visible properties of those elements, and the relationships among them [7]. Its tasks are still hard to accomplish, demanding forceful effort and time. The problem of creating a software architecture model is similar to solving any other problem, where, iteratively, we understand the problem, find a solution, and evaluate the final result. As building software architecture is costly, the activities to its creation are not formally performed in agile software development contexts, where software architectures are built, but with little planning and analysis. Our approach uses mind maps to create essential software artifacts in a cognitive and interactive way and model-driven techniques to automatically generate artifacts (e.g., a reference architecture), reducing the modeling process effort.

URI: http://hdl.handle.net/10125/50569
ISBN: 978-0-9981331-1-9
(CC BY-NC-ND 4.0)

2.2. Agile development

Agile development aims at reducing the effort-intensive tasks in software development, focusing on fast response to the various changes in a project [8]. The Agile Manifesto establishes values and principles to guide the agile development [4]. In recent years, researchers and practitioners have proposed several agile practices [9], which have been catalogued by the agile alliance in its "subway map to agile practices" [10]. The following topics show the meaning of Agile Practices (AP#) used in our method [10]:

[AP01] Iterations: is a timebox during which development takes place.
[AP02] User stories: are functional increments describing what must be developed by the team.
[AP03] Facilitation: is any action that facilitates the development.
[AP04] Team: is a small group of people, assigned to the same project.
[AP05] Backlog: is an ordered list of items representing everything that may be needed to deliver a specific outcome.
[AP06] Iterative development: is the "repetition" of software development activities for potentially "revisiting" the same work products.
[AP07] Incremental development: is the adding of user-visible functionality to the previous software version.
[AP08] Ubiquitous language: is the use of the vocabulary of a given business domain, not only in discussions about the requirements for a software product, but also in discussions of design.
[AP09] Simple design: is the design that uses the practice often reduced to the acronym YAGNI (You Aren't Gonna Need It).

2.3. Cognitive maps

A "*cognitive map is a mental device and store which helps to simplify, code and order the endlessly complex world of human interaction with the environment*" [11]. A mind map is a type of cognitive map used to view, classify and organize concepts, to generate new ideas in a straightforward and intuitive way, to emphasize relevant keywords, and to associate elements in branches [12]. A mind map is composed of a central node (representing the main concept of model) and ramifications of topics and sub-topics from the central node. We use a mind map to structure business and software models to facilitate knowledge transfer from business to software [27].

2.4. Business value

A business model is a lightweight, semi-formal and conceptual technique, inspired in business science, requirements engineering and conceptual modeling to model business ideas [13]. Its main goal is to identify *who* is offering *what* to *whom* and expects *what* in return. (Note the difference with a business process model (e.g., BPMN [14]) that describes *how* processes should be carried out, and by *whom* [15].) The central notion in a business model is the concept of *value*, to explain the creation, addition, and the exchange of value between stakeholders [15].

A *value model* shows how a business value is created and exchanged in an inter-organizational network, aiding detecting business opportunities [16]. *Value* is the reason why people and companies trade with each other, offering money to get something in return. So, a value model represents a business model from an economic perspective, and determines the exchanged economic values and their intervenients [16]. Its alignment with software development is critical to meet the customers' satisfaction.

RAMA uses DVD (Dynamic Value Description language) [17] to specify business values. DVD has proved to be an ease to use, useful, effective and efficient value-driven approach [18]. It offers an environment where stakeholders can share their values exchanged views in a semi-structured and concise mind map model (Section 2.3). Figure 1 shows a DVD model for an abstract shop business, structured as a mind map.

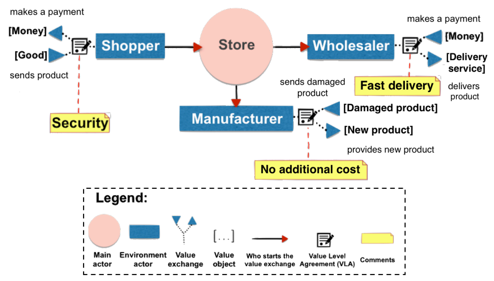


Figure 1. DVD model example.

Actors are environment entities economically independent. The business analysts focus defines the *main actor* (central node of the model), and their focus change along the specification process. Each time they focus on one actor (the main actor), identify its relationship with other *environment actors*, creating an inter-organizational network. From each such relationship, a *value exchange* (transfer of resources) is defined, showing economic reciprocity through two value ports (blue arrows connected to value exchanges in Figure 1 pointing to *value objects* such as money and good). The arrows direction is set based on the environment actor. Each value exchange has a textual description which is not represented in the visual model. Next, we define *who starts* the value exchanges through a configuration of arrows (in red) between the main actor and the environment actors, helping understanding the model. For example, `Shopper` starts a value exchange with `Store`, by paying `Money` in exchange for a `Good`.

When focusing on one actor, the supporting tool displays it, dynamically, as the central node of the model. Each value exchange requires a *value level of agreement* (VLA) between the actors involved, which refers to the minimal business rule agreed among them with no clear-cut satisfaction criteria. In the example, the shopping transaction between `Shopper` and `Store` must be secure, leading to add `Security` to the corresponding value exchange.

2.5. Model driven engineering

Model-Driven Engineering (MDE) automates repetitive and error-prone tasks through an automatic processing model aiming at reducing the accidental complexity involved in software development [19]. It has been successfully used in industries, including telecommunication, automotive, aerospace, and business information systems [20]. MDE focuses on abstracting the details of a complex problem, concentrating developers on producing top-level abstraction models to generate complex software artifacts automatically. Hence, MDE uses models as first class entities, aiming at increasing productivity, augmenting interoperability, and facilitating communication [21], [22].

Developing software from models requires these to be rigorously defined [23], what is achieved through metamodels and automatic transformations [24]. Metamodels are used to implement model transformations and to create DSLs (Domain Specific Languages). Model transformations incrementally and automatically refine, refactor or re-engineer abstract source models [25] until producing a solution model (known as *Model to Model*, or M2M, and *Model to Code*, M2C). DSLs are languages designed to be useful for specific sets of tasks and particular domains [26], realize particular points of view of a problem, and create rigorous modeling editors.

Our approach uses MDE to generate software artifacts, like a reference architecture model.

3. The RAMA method

The RAMA macro process in Figure 2 has six activities: *specify value model*, *prioritize value exchanges*, *specify conceptual models*, *identify concepts overlaps*, *decision analysis*, and *create reference architecture*. To *specify a value model*, the business person creates a DVD model in a meeting session with the participation of the development team (related to the AP04 practice). Then, the business *prioritizes value exchanges*, according to the business return on investment (ROI).

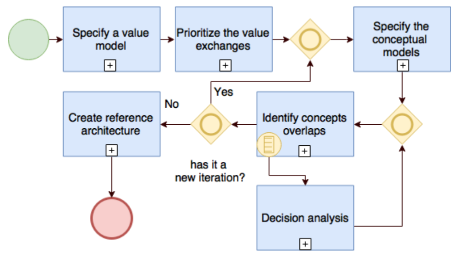

Specify a value model
Prioritize the value exchanges
Specify the conceptual models
Yes
No
Create reference architecture
Identify concepts overlaps
has it a new iteration?
Decision analysis

Figure 2. RAMA's macro-process.

Next, the business person and the development team, with the help of a facilitator (agile practice AP03), *specify conceptual models*, for the value exchanges using mind maps (AP08). After, the development team *identifies concepts overlaps* among conceptual models. When an overlap is found, the team decides (*decision analysis*) which of the models take the responsibility for that concept. These activities are performed iteratively and incrementally (AP06 and AP07). Finally, model-driven techniques are used to *generate a reference architecture* with its architectural components and relationships.

3.1. Specify value model

The business person explains to the development team how the business works, by representing business values, what results in a DVD value model. This model is created as follows: (1) *Specify main actor* by representing the focus of the analysis (the business for which the information system will be developed or evolved). In the example of Figure 3, `Store` is the central node. (2) *Identify environment actors* that directly interact with the main actor. The result is adding `Shopper` and `Manufacturer` (who changes products damaged) and `Wholesaler` (who offers a fast delivery service) to Figure 3. (3) *Set value exchanges* defines the value elements related to each value port. For example, `Shopper` gives `Money` to `Store` in exchange for a `Good`. `Money` is depicted in the value exchange's output port (arrow heading out) and `Good` in the input port (arrow heading in).

(4) *Set who triggers each value exchange* identifies the actor causing the value exchange. For instance, `Shopper` starts by *making a payment*. (5) *Set value level agreement* defines the contracts, or restrictions, for the value exchange. In the example, the shopping transaction must be secure, leading to add `Security` to the value exchange between `Shopper` and `Store`.

3.2. Prioritize value exchanges

Priorities are given in two rounds. For the first, the business person uses a scale (high, medium, and low) to define the priority of each value exchange according to ROI. The DVD model uses a color code to represent each scaling value visually (red, yellow and blue, respectively). These priorities guide the development iterations (AP01), where the highest ranked will be implemented first. Figure 3 shows these priorities and the context menu to choose them.

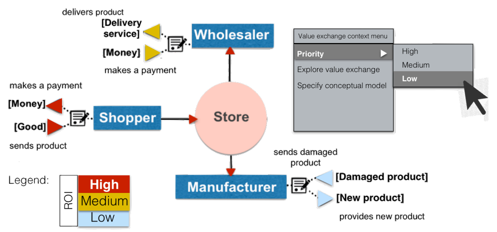


Figure 3. 1st-round priority to value exchanges.

Next, the business person and development team defines user stories (AP02) for each value exchange (or only those with higher priority contained in an iteration). User stories describe software requirements aligned with the business values. The second prioritization round, done by the development team, happens for value exchanges with the same priority (and user stories already described). The goal is to distinguish the value exchanges with the same priority, solving potential future conflicts. Then, it is clear which value exchanges must be handled first during the information system development.

3.3. Specify conceptual models

Development team creates conceptual models to (or part of) the value exchanges. To aid visualization and assure traceability between value exchanges and respective conceptual models, a behavior tree view is generated from the DVD model using M2M transformations (Figure 4, generated automatically cf. Section 4.1.1). The top level is the parent node (backlog, related to AP05), each of the three nodes in the second level represent a priority value from the priority scale used in the first-round, the third level has the value exchanges (each number represents the second-round priority), and the fourth level has the user stories.

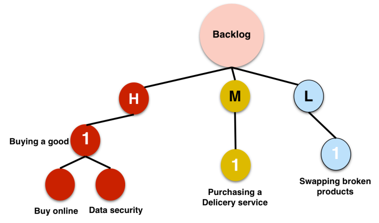


Figure 4. Behavior tree view.

The conceptual modeling activity is collaborative, involving the business person, the development team, and a facilitator (who can be a member of the team) [27]. Armed with the value exchange specification (composed of a set of user stories), the facilitator helps the business person and the development team build the conceptual models, and uses a mind map to answer questions like: (i) What is the central concept of the problem domain? (ii) What are the sub-concepts directly related to the central concept and that are relevant to the system? (iii) What data must be managed and stored? These help eliciting relevant responses from the business person to build the mind map model [27], which is used to aid communication. It diminishes the semantic gap between the business person and the development team [28], [29].

From the conceptual models structured as mind maps, the development team creates class diagrams using M2M transformations [29]. For example, Figure 5-a shows a conceptual map for "security data" user story (where "profile" was elected as the main term in the user story) and Figure 5-b (cf. Section 4.1.2) shows the initial class diagram generated from this mind map.

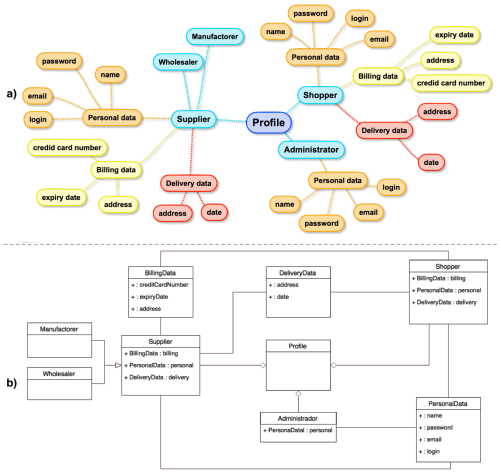


Figure 5. Conceptual model structured as (a) a mind map and (b) a class diagram.

Now the development team, owning IT background, adds detail to the class diagram by defining data types (e.g., String, Integer), access modifiers to attributes (e.g., public, private), cardinalities, and methods names [29].

3.4. Identify concept overlaps

The development team searches for similar concepts among conceptual models, identifying possible overlaps. Figure 6 shows an example of a representation of a conceptual overlap.

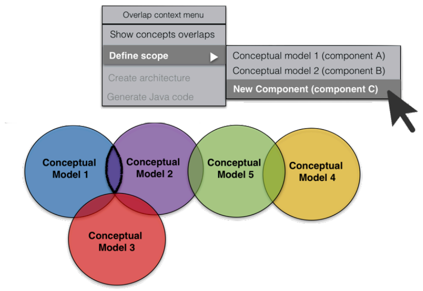


Figure 6. Conceptual overlaps and decision analysis.

3.5. Decision analysis

The development team must decide where the overlapped concepts belong, following the principles of domain-driven design [30], where the domain is modularized with a concise set of concepts guiding the software structure. A software component encapsulates a cohesive set of system's functionalities. Those functionalities handle a set of entities. Those entities are represented as concepts in a conceptual model. Then when defining the conceptual model boundaries we also define the software component boundaries, leading to each conceptual model to "map" to a software component in our reference architecture. So, the development team must decide if the overlapped concepts (Figure 7) belong exclusively to component A (matching to conceptual model 1, for example) or component B (matching to conceptual model 2) or neither (a new component C). Figure 6 shows this decision analysis, when the development team chooses that the overlap should form a new component.

3.6. Generate reference architecture

After defining the scope of each component, the development team generates the reference architecture model and names of each component. The relations between components are detected through the conceptual models overlaps: if the *conceptual model A* uses a concept from *conceptual model B*, then they are related. Figure 7 shows the resulting architecture model, which is automatically generated using MDE techniques. If class diagrams created by the activity *specify conceptual models* were completed to have methods, then the architectural components could be generated with their interfaces (it is also possible to generate a skeleton of Java code). Otherwise, if the methods were not specified in the class diagrams, developers must define the interfaces manually.

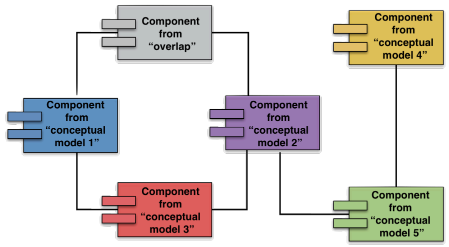

Component from "overlap"
Component from "conceptual model 4"
Component from "conceptual model 1"
Component from "conceptual model 2"
Component from "conceptual model 3"
Component from "conceptual model 5"

Figure 7. Reference architecture example.

4. Evaluating RAMA

RAMA's evaluation is in two parts: (i) checking the method's feasibility by building proof-of-concept tools to support the most important parts of the method (business value modeling, conceptual modeling, identification of concepts overlaps, and support decision analysis), and (ii) applying the method to create a reference architecture model for an industrial online auction system in an agile development. This reference architecture model was the result of the first sprint of a software development process based on the Scrum which had a development team composed of only two people.

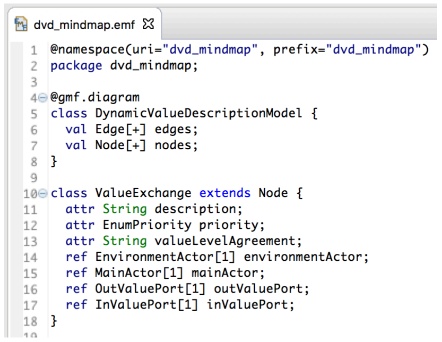
4.1. Proof-of-concept tools (first evaluation)

We identified three points for automation and build a supporting tool to each point: (i) a DSL to create the DVD model; (ii) a DSL to create the conceptual models; (iii) an algorithm to identify overlaps and support to decision analysis.

4.1.1 DSL for value modeling

A DVD editor was implemented using the Eclipse EuGENia tool [31], allowing the creation of models syntactically validated. EuGENia automatically generates the background models needed to implement a GMF editor from a single annotated

Ecore metamodel (Enfatic model). Figure 8 shows a fragment of the Enfatic model used in our tool. The editor allows the modeler to create a DVD model dragging DVD's objects contained in a palette box. The implemented transformation rules used the Epsilon Transformation Language (ETL) to generate the behavior tree view (Figure 4) automatically.



```
@namespace(uri="dvd_mindmap", prefix="dvd_mindmap")
package dvd_mindmap;

@gmf.diagram
class DynamicValueDescriptionModel {
  val Edge[+] edges;
  val Node[+] nodes;
}

class ValueExchange extends Node {
  attr String description;
  attr EnumPriority priority;
  attr String valueLevelAgreement;
  ref EnvironmentActor[1] environmentActor;
  ref MainActor[1] mainActor;
  ref OutValuePort[1] outValuePort;
  ref InValuePort[1] inValuePort;
}
```

Figure 8. Fragment of Enfatic model.

4.1.2 DSL for conceptual modeling

The DSL infrastructure for conceptual modeling consists of (i) a MindMappingModeler component a (ii) DomainModelExtractor component, and (iii) a DomainModelTool component. Figure 9 shows the infrastructure of the DSL. Through the MindMappingModeler, the business person and the development team create the mind-map-like conceptual model structure. The DomainModelExtractor component performs processing (binding) and produces the UML class diagrams representing the domain. These class diagrams artifacts are the input to the DomainModelTool (an editor of class diagrams).

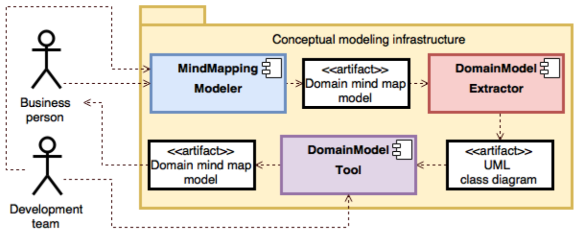

Conceptual modeling infrastructure
MindMapping Modeler
<<artifact>> Domain mind map model
DomainModel Extractor
Business person
<<artifact>> Domain mind map model
DomainModel Tool
<<artifact>> UML class diagram
Development team

Figure 9. Tool infrastructure.

4.1.3 Overlap identification

To facilitate the overlap identification, we used the well-known Levenshtein distance algorithm [32], which measures the edition distance between two words, calculating how many operations it needs to transform a word source in another word target. There is a significant probability of the concepts to be the same when the edition distance between the words is short. The similarity detection is a semi-automatic process where the algorithm detects the nodes with similar names (not necessarily the same) and solicits a confirmation (is the concept "x" in the mind map "m" analogous to the concept "x1" in the mind map "m1"?). If the person confirms the similarity, then the algorithm registers the concept overlap. Levenshtein distance algorithm was enough to the proof-of-concept evaluation, but we believe that future research must be performed in natural processing languages algorithms (e.g., identification of synonymous words).

4.2. Case Study (second evaluation)

This is an industrial online auction system that is part of a Brazilian gas station chain fidelity program and was performed in an agile development.

4.2.1. Business description of the online auction

When a gas station chain customer registers in the system, he earns 50 coins to bet in any auction of the system (each coin allows one bet). Additional coins are acquired if a customer (i) shops in a gas station (receives the product and coins), (ii) wins an auction (places a bet and expects to be the winner), or buys coins packages (pays for coins). The system provides several auctions concurrently, always selling cheaper than market price. The idea is not to earn by selling a third (partner) company's product or service (i.e. goods) but by having a large number of bets or selling its own goods.

An auction starts with a minimum, current and maximum price of goods, a start time, and an envisaged end time. It begins with the minimum price and, each time a bet is placed, the current price is increased by R$ 0,01. If the auction finalizes before reaching the maximum price, the customer makes a very good acquisition (paying much less than market price). If the price reaches the maximum price, he is still acquiring the good cheaper than in the market. Thirty seconds from the deadline, new bet delays the finish time in thirty seconds, allowing time for more bets. The winner is the owner of the last bet, who is contacted by e-mail and has thirty days to pay with a credit card for the good acquired. The credit card company must provide a secure financial service in exchange of a payment. After the payment is confirmed, the gas station uses a delivery service to deliver the product to the customer. If payment is not concluded, the gas station chain creates a new auction with that same product. The online auction system of the gas station chain sells advertisement, earning goods to be auctioned in exchange for publicity in their own website (large number of visualizations).

4.2.2. Applying RAMA

Specify a value model. Figure 10 shows a DVD model representing the business value exchanges for the online auction business. First, we identified `Online auction`, `Customers`, `Partners`, `Credit card company`, and `Delivery company` as the actors of this business. Online auction is the focus of the analysis (*main actor*) and the other actors are those with whom value exchanges occur (*environment actors*).

The four value exchanges with `Customers` are (i) register their data in the system, (ii) buy in the gas station, (iii) place bets, (iv) pay for product won at auction. In these value exchanges, `customers` start the actions. `Partners'` offer goods to be auctioned and start the action. The `Credit card company` offers a financial service contracted by the `Online auction` that starts the action. The `Delivery company` offers the delivery service to deliver goods to the winners, and the `Online auction` starts the action. The VLAs for the value exchanges with `Customer` are *free coins* and is *low cost* of the good, with `Partners` is *large number of visualizations*, with `Credit card company` is *Security*, and with `Delivery company` is *fast* (see Figure 10).

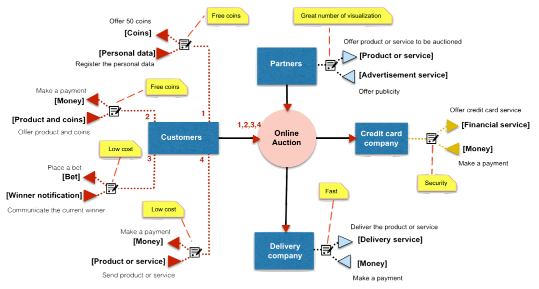


Figure 10. DVD model for the online action.

Prioritize value exchanges. The prioritization of value exchanges are done based on ROI. Thus, the business person sets the customers' value exchanges priority "high", the financial value exchange as "medium", and the remaining value exchanges as "low" (Figure 10). Once the high-level prioritization is done, the business person specifies user stories for the value exchanges with high priority. Table 1 describes some of these user stories.

Specify conceptual models. Figure 11 shows the conceptual model represented as mind maps for VE1. The central term is "*account*". Customer and partner have accounts. Customer registers mandatory data, personal data, billing data, delivery data and has a wallet. The wallet knows the customer's quantity of coins and holds transactions history. The central node of VE2 is "*coins acquirement*" (Figure 12). When a customer buys a good in the gas station, an invoice is issued. *Customer* must inform the data of the *invoice* so that the system performs a calculation (*conversion table*) of how many coins will be added in the customer's wallet. For VE3, the central node is "*bet*" (Figure 13). *Customer* places a bet in an *auction.* The auction offers an *object* (service or product) and saves the history of all the bets. When the auction finishes, a *notification* is sent to the winning customer.

Table 1. User stories.

ID	Value exchange	User story
VE1	Registering data	As a customer, I want to create an account so that I can place a bet. As a customer, I want to receive free coins so that I can place bets in auctions.
VE2	Buying in the gas station	As a customer, I want to earn free coins when I buy goods in the gas station so that I can place bets in the auctions.
VE3	Placing a bet	As a customer, I want to see all auctions available so that I can choose where I will place my bets. As a customer, I want to place a bet on an auction so that I can be the winner. As a customer, I want to know if I am losing an auction where I am betting so that I can place another bet. As a customer, I want to know when the auction will finalize so that I can place another bet if I am losing the auction. As a customer, I want to be notified when I am the winner of an auction so that I can conclude and pay.
VE4	Buying the auctioned good	As a customer, I want to know the price of the good I won so that I can pay for it. As a customer, I want to know the deadline to confirm the payment so that I can perform the payment on time.

For VE4, the central node is "*payment*" (Figure 14). *Customer* knows the price of the *auction* s/he won and offers *billing data* to complete the payment process. During this process, a *monitor* checks that all payment steps (e.g., if the payment was performed before the expiry date) and all changes that may happen during the process are registered (*history*). When payment is concluded and the *object* auctioned is of type *product*, then the *delivery* process starts.

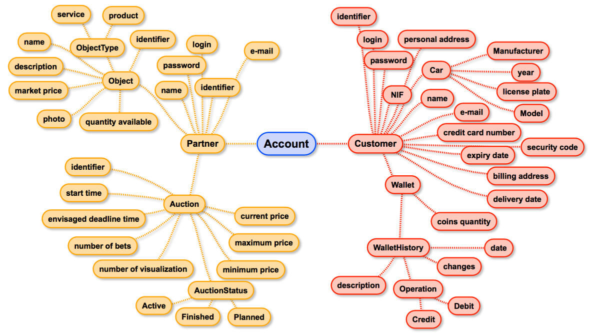


Figure 11. “Registering data” conceptual model.

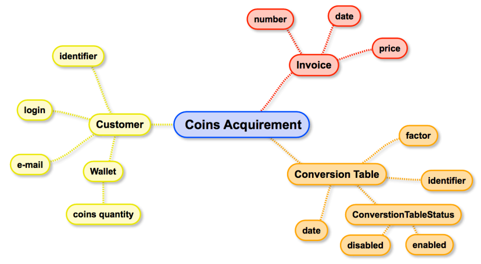


Figure 12. “Buying” conceptual model.

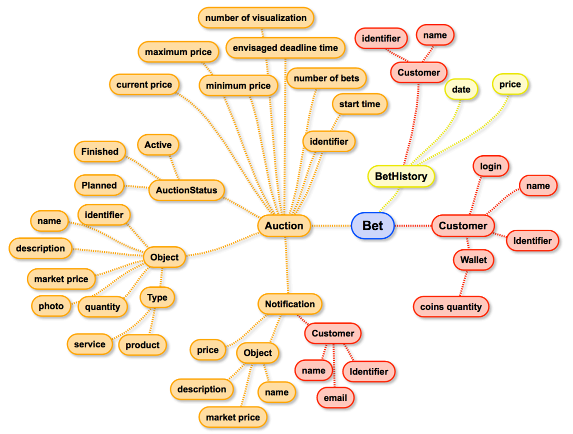


Figure 13. “Placing a bet” conceptual model.

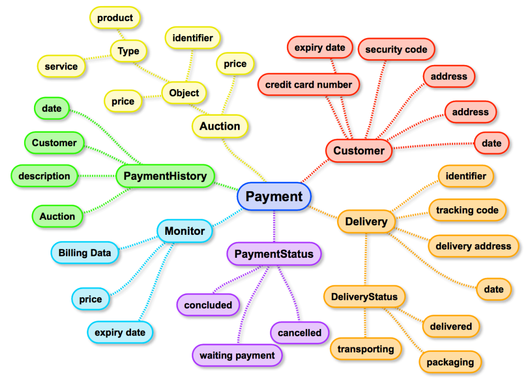


Figure 14. “Payment” conceptual model.

From these conceptual models, the development team can generate class diagrams and complete them with additional details. All class diagrams can be found in a zip file at https://goo.gl/7PrXZ4.

Identify concepts overlaps & decision analysis. A total of eight overlaps were found during the identification task. Table 2 shows the overlaps and the decisions made.

Table 2. Overlaps

#	Conceptual model A	Conceptual model B	Overlap concept	Decision analysis
1	Registering data	Buying in the gas station	Customer	Create new “Customer” component
2	Registering data	Placing a bet	Auction	Create new “Auction” component
3	Registering data	Placing a bet	Object	Create new “Object” component
4	Buying in the gas station	Customer	Customer	Customer
5	Placing a bet	Customer	Customer	Customer CRUD
6	Placing a bet	Payment	Auction	Placing a bet
7	Placing a bet	Object	Object	Object
8	Payment	Customer	Customer	Customer

Generate reference architecture. To finalize the process of combining architecture design and agile practices, the development team automatically generates the reference architecture and names each component (AP09). Figure 15 shows the reference architecture generated where the “*registering data”* was renamed to “*Partner”, and “placing a bet”* to “*Bet”*. The generated components are complemented with the interfaces (offered and required) according to methods specification in the class diagrams. The more information the class diagrams have (e.g., methods, signatures, attribute types) the more complete the reference architecture generated is in terms of interfaces. Component relationships are detected automatically through the concepts overlaps.

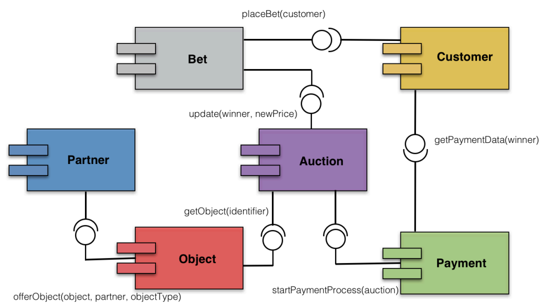


Figure 15. Reference architecture for the online auction system.

4.2.3. Discussion of the second evaluation part

We used value modeling, domain-driven, model-driven, and conceptual modeling techniques together with agile practices to produce a feasible approach to combine software architecture design and agile development. Given that several parts of the process are generated automatically, we reduced time to create the various models, aligned with the initial business values. However, the granularity of the reference architecture created is directly related to the granularity of the conceptual models specifications. The more detailed the specification is, the more detailed the reference architecture generated is. For instance, if detailed class diagrams were created during conceptual modeling, the generated reference architecture could show the components' interfaces. The result shows that RAMA helps creating a software architecture for an information system in an agile software development context. The next step is to validate these results with an empirical experiment with Agile teams.

5. Related work

Our method differs from the other existing methods (e.g., [33], [34], [37], [38]), as it uses DVD to represent business values and uses user stories to bridge business value representation and software requirements specification (conceptual modeling).

Regarding the architectural description process in agile development, the product backlog is the most important artifact used as input [3]. In general, the product backlog is used as a knowledge base to create an architectural document [35]. Our method uses conceptual models instead of product backlog as the knowledge base because they have a lot more details of the entities and attributes of the system. Also, the conceptual model used in our method is structured through mind map, aiming at decreasing the effort and improving the business understanding during its building. Also, the most frequently used tools for architectural description are an office white board and an online user-editable Wiki [33], [36]. Our method uses tools to generate and handling models using model-driven techniques, aiming at reducing error-prone activities and time to build a reference architecture.

6. Conclusion and future work

This paper shows a value-centric development method to Reference Architecture Modeling in an Agile software development (RAMA) context, addressing the lack of techniques (approaches, tools, methods) to support the architecture-agility combination. The proposed method uses model-driven techniques to create a reference architecture for an information system aligned with the business values. We evaluated the method by applying it to an industrial case study and developing proof-of-concept tools to check the feasibility of automating parts of the process, hence contributing to make the agility-architecture combination a lightweight process. The results show that RAMA enables the creation of a software architecture model for an information system in an agile software development context. We plan to build an integrated environment for the proof-of-concept tools and improve the conceptual overlaps algorithm to identify synonymous words, and set up an empirical experiment with agile teams to confirm our results.

6. References

[1] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and Architecture: Can They Coexist?," IEEE Software, 27(2), 2010.

[2] S. Freudenberg and H. Sharp, "The Top 10 Burning Research Questions from Practitioners," IEEE Software, 27(5), 2010.

[3] C. Yang, P. Liang, and P. Avgeriou, "A systematic mapping study on the combination of software architecture and agile development," The Journal of Systems and Software, vol. 111, 2016.

[4] M. Fowler and J. Highsmith, "The Agile Manifesto," Software Development, vol. 9, 2001.

[5] L. Hohmann, Beyond Software Architecture: Creating and Sustaining Winning Solutions, 1st ed. 2003.

[6] D. Garlan, "Software architecture: a travelogue," presented at the Proceedings of the on Future of Software Engineering, New York, USA, 2014.

[7] L. Bass, P. C. Clements, and R. Kazman, Software architecture in practice, 2nd ed., Addison-Wesley, 2003.

[8] J. Erickson, K. Lyytinen, and K. Siau, "Agile modeling, agile software development, and extreme programming: the state of research," Journal of Database Management, 16(4), 2005.

[9] J. Medeiros, A. Vasconcelos, M. Goulão, and C. Silva, "An approach based on design practices to specify requirements in agile projects," ACM SIGAPP Symposium On Applied Computing, 2017.

[10] Agile Alliance, “Subway map to agile practices,” https://www.agilealliance.org/agile101/subway-map-to-agile-practices/.

[11] D. J. Walmsley, T. F. Saarinen, and C. L. MacCabe, “Down under or centre stage? The world images of Australian students,” Australian Geographer, 21(2), 1990.

[12] T. Buzan and B. Buzan, The Mind Map Book. Pearson Education, 2006.

[13] A. Rasiwasia, “Meta Model for Business Model Design: Designing a Meta model for E3 value model based on MOF”, MSc thesis, Sweden, 2013.

[14] S. A. White, “Introduction to BPMN,” IBM Cooperation 2.0, 2004.

[15] J. Gordijn, H. Akkermans, and H. V. Vliet, “Business Modelling Is Not Process Modelling,” in International Conference on Conceptual Modeling, 1921(5), Springer Berlin Heidelberg, 2001.

[16] J. Gordijn, “Value-based Requirements Engineering,” PhD Thesis, 2002.

[17] E. Souza, S. Abrahao, A. Moreira, J. Araújo, and E. Insfran, “Comparing Value-Driven Methods: an experiment design,” presented at the 2nd International Workshop on Human Factors in Modeling, Saint Malo, France, 2016.

[18] E. Souza, A. Moreira, J. Araújo, S. Abrahao, and E. Insfran, “Evaluating the efficacy of value-driven methods: a controlled experiment”, 26th International Conference on Information System Development, Larnaca, Cyprus, 2017.

[19] D. S. Kolovos et al., “Model Driven Grant Proposal Engineering,” in International Conference on Model Driven Engineering Languages and Systems, Springer International Publishing, 2014.

[20] G. Mussbacher et al., “The Relevance of Model-Driven Engineering Thirty Years from Now”, in International Conference on Model Driven Engineering Languages and Systems, Springer International Publishing, 2014.

[21] D. C. Schmidt, “Guest Editor's Introduction: Model-Driven Engineering,” IEEE, 39(2), 2006.

[22] J. P. A. Almeida, “Model-Driven Design of Distributed Applications”, in OTM Confederated International Conferences" On the Move to Meaningful Internet Systems, Springer Berlin Heidelberg, 2004.

[23] Y. Singh and M. Sood, “Model Driven Architecture: A Perspective”, presented at the Advance Computing Conference, 2009. IACC 2009. IEEE International, 2009.

[24] A. G. Kleppe, J. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[25] F. Wanderley and J. Araújo, “Generating goal-oriented models from creative requirements using model driven engineering”, International Workshop on Model-Driven Requirements Engineering (MoDRE), 2013.

[26] R. C. Gronback, Eclipse modeling project: a domain-specific language (DSL) toolkit. Addison-Wesley, 2009.

[27] F. Wanderley and D. S. da Silveira, “A Framework to Diminish the Gap between the Business Specialist and the Software Designer”, presented at the International Conference on the Quality of Information and Communications Technology, 2012.

[28] S. Robertson and J. Robertson, Mastering the Requirements Process: Getting Requirements Right, 3rd ed. Addison-Wesley Professional, 2012.

[29] S. Ambler, Agile modeling: effective practices for extreme programming and the unified process. John Wiley & Sons, Inc., New York, 2002.

[30] E. Evans, Domain-driven design: tackling complexity in the heart of software. Boston: Addison-Wesley, 2003.

[31] D. Kolovos, “EuGENia website”, http://www.eclipse.org/epsilon/doc/eugenia/. [Accessed: 08-Jun-2017].

[32] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval, vol. 1. Cambridge University Press, 2008.

[33] J. Sauer, “Architecture-Centric Development in Globally Distributed Projects,” in Agility Across Time and Space, no. 22, Springer Berlin Heidelberg, 2010.

[34] R. L. Nord, I. Ozkaya, and R. S. Sangwan, “Making Architecture Visible to Improve Flow Management in Lean Software Development,” IEEE Software, 29(5), 2012.

[35] V.-P. Eloranta and K. Koskimies, “Aligning architecture knowledge management with Scrum”, presented at the the WICSA/ECSA 2012 Companion Volume, ACM, 2012.

[36] R. L. Nord and J. E. Tomayko, “Software architecture-centric methods and agile development,” IEEE Software, 23(2), 2006.